Prozedurale Level-Generierung

Masterarbeit

zur Erlangung des akademischen Grades Master of Arts

Vorgelegt von Fabian Utsch

Master Studiengang Zeitabhängige Medien – Games
Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien, und Information
Department Medientechnik

Erstprüfer: Prof. Dr.-Ing. Boris Tolg

Zweitprüfer: Prof. Ralf Hebecker

Fabian Utsch

Titel der Master-Thesis / Title of the Master Thesis

Prozedurale Level-Generierung

Procedural Level Generation

Stichworte / Keywords

Prozedural, Level, Spiele, Houdini, Tomb Raider

Procedural, Level, Games, Houdini, Tomb Raider

Zusammenfassung / Abstract

Der Begriff *Prozedural* ist in der Games-Branche mittlerweile ein Synonym für grenzenlose Inhalte geworden. Prozedural steht dabei für mehr als nur eine unerschöpfliche Quelle von Inhalten. Am Anfang dieser Arbeit werden daher zunächst die Vielseitigkeit des Begriffs im Hinblick auf die Definition innerhalb der Games-Branche und die unterschiedlichen Anwendungsgebiete betrachtet.

Anschließend wird ein prozedurales System entwickelt, welches es sich zur Aufgabe macht, ein Level auf Basis bereits vorhandener Geometrie zu generieren. Dazu wird der Spieleklassiker *Tomb Raider* (Teil 1) verwendet und die vorhandenen Geometriedaten werden in ein zeitgenössisches Level transferiert.

The term procedural has become a synonym for limitless content throughout the games industry. It stands for more than just an unlimited source of content. The beginning of this thesis will attempt to shed some light on the diversity of the term itself, considering the definition of it throughout the games industry and the various application areas.

Subsequently, a procedural system is developed, which task it to generate a level on the basis of already existing geometry. The game classic *Tomb Raider* (part 1) is used and the existing geometry data is transferred to a contemporary level.

Inhaltsverzeichnis

1 Einführung

Seit Jahrzehnten sind Computer im täglichen Leben nahezu überall im Einsatz. Sie erleichtern den Alltag, indem sie den Benutzern bei Aufgaben unterstützen oder ihm diese gänzlich abnehmen.

Auch Bereiche die einer analogen Herkunft entspringen, wie beispielsweise die Kunst, wurden stark durch die Entwicklung des Computers beeinflusst. Künstler*innen und Designer*innen sind stets dazu geneigt, ihre bestehenden Arbeitsweisen in die digitale Welt zu transferieren[1]. Die digitalen Tools sollen sich dabei möglichst nah an ihren analogen Verwandten orientieren. Hierbei werden Arbeitsschritte vereinfacht und um digitale Vorteile ergänzt. Dabei wird jedoch häufig außer Acht gelassen, was der Computer außerdem für uns leisten kann.

In manchen Bereichen des Designs, zumeist in der Architektur und im Produktdesign [1], dient der Computer nicht nur als Werkzeug, sondern liefert auch Lösungsansätze und hilfreiche Vorschläge. Diese Arbeitsweise wird zumeist als prozedurales Arbeiten bezeichnet oder mit dem Begriff des generativen Designs in Verbindung gebracht.

Bei der Erstellung von digitalen Inhalten, egal ob im Hinblick auf Film, Werbung oder Videospielen, ist dieser Ansatz nicht unbekannt. Er wird jedoch nur von wenigen Künstler*innen und Designer*innen genutzt oder in begrenzter Komplexität angewandt.

Beim generativen Design oder der prozeduralen Gestaltung wird der Designer oder die Designerin beim Erstellungsprozess von einem Algorithmus begleitet. Das Ziel ist es ein System zu generieren, welches Arbeitsschritte auf den Computer verlagert und eine Gestaltung unter festgelegten Regeln anwendet [2]. Der Künstler*innen arbeiten fortan nicht nur am Computer, sondern mit dem Computer, indem sie aus Vorschlägen auswählt oder Arbeitsschritte automatisiert.

---

[1] Im Grafikdesign als Skeuomorphismus bezeichnet: https://www.interaction-design.org/literature/topics/skeuomorphism

Die Automatisierung von Arbeitsabläufen oder Lösungsvorschlägen ist auch in der Games-Branche kein neuer Gedanke [3]. Oft damit in Verbindung gebracht wird der Titel *Rogue*, der bereits 1980 erschien ist. Bei diesem Titel begeben sich Spieler*innen in ein Verlies um ein Amulett zu finden, wobei mehrfach verschiedene Gegner erscheinen. Der Reiz besteht unter anderem darin, dass Spieler*innen nie wissen, was sie erwartet. Die Level-Segmente werden prozedural generiert, so dass keine Spielrunde der vorherigen gleicht. Ebenso war die prozedurale Generierung eine Lösung für die damals begrenzte Speicherkapazität. Levels belegten nun nicht unnötig viel Festplattenspeicher, da neue Levels nur dann generiert wurden, wenn es nötig war.

Moderne Computer besitzen Festplattenspeicher im Überfluss, doch die Ansprüche an die Spielwelten sind gewachsen. Wir haben uns aus dem Verlies begeben und erforschen Inseln[2], besuchen fantastische Welten[3], oder kämpfen uns durch Großstädte[4]. Jedes Jahr wird ein neuer Titel angekündigt, der eine noch größere Spielwelt bietet als je zuvor. Diese muss einem immer größeren Verlangen nach noch realistischeren und detaillierten Leveln gerecht werden.

Um diesen Anforderungen in Anbetracht von Budget und Produktionszeit gerecht zu werden, müssen bestehende Workflows überdacht werden [4].

Die prozedurale Generierung von Inhalten ist eine Möglichkeit diese Anforderungen zu meistern, denn so werden Arbeitsabläufe weitestgehend automatisiert. Dies ermöglicht eine neue Arbeitsweise und schafft Werkzeuge, mit deren Hilfe weitläufige und detailverliebte Welten entstehen können[5].

---

2 Far Cry: https://far-cry.ubisoft.com/game/de-de/home

3 The Elders Scrolls V: Skyrim: https://elderscrolls.bethesda.net/de/skyrim

4 GTA 5: https://www.rockstargames.com/V/

5 GDC Talk zur Erstellung von Manhattan in Marvel's Spider-Man: https://www.gamasutra.com/view/news/341095/Video_Procedurally_creating_Manhattan_for_Marvels_SpiderMan.php

1.1 Aufbau dieser Arbeit

Was hat es mit dem Begriff *prozedural* auf sich und was ist damit in der Games-Branche gemeint? Der Begriff wird unter Programmierer*innen und Designer*innen in den letzten Jahren nahezu inflationär verwendet. Auf welche Weise dieser Begriff zwei unterschiedliche Fachgebiete miteinander verbindet und wie es dazu kam, soll im theoretischen Teil dieser Arbeit behandelt werden.

Im zweiten Teil liegt der Fokus auf der Anwendung prozeduraler Gestaltung. Ziel der Arbeit ist die Entwicklung eines Systems, welches weite Teile der Level-Generierung übernimmt und es den Anwender*innen somit ermöglicht, sich auf das Gameplay im Level zu konzentrieren. Es folgt eine Dokumentation, welche die Prinzipien abbildet, die als Basis der zu bewältigenden Aufgabenstellungen entwickelt wurden und schließlich das Ergebnis, das unter der Anwendung entstanden ist

2 Grundlagen

2.1 Procedural & Proceduralism

Der Begriff *Prozedural* (Englisch: procedural) wird in verschiedenen Zusammenhängen und Themengebieten genannt, wie z.B. in der Programmierung, dem Design, Gamedesign oder der künstlichen Intelligenz, um nur wenige zu nennen. Um eine präzisere Definition herauszuarbeiten, beschränkt sich der Begriff in diesem Fall einzig auf die Verwendung prozeduraler Gestaltung im Kontext von Games:

> *„Procedural content generation (PCG) in games refers to the creation of game content automatically using algorithms." [5]*

Innerhalb des Algorithmus werden Vorgehensweisen festgelegt, die bei der Lösung von Problemen ihre Anwendung finden. Am Ende dieser Einzelschritte steht das Ergebnis, in Form eines Objektes, einer Animation oder anderer Inhalte innerhalb des Games.

Anstelle des Begriffs *Algorithmus* wird unter anderem auch der Begriff *System* verwendet. In diesem Zusammenhang ist das „prozedurales System" gemeint. Dieses besteht aus einem Satz von Regeln, der eine Autonomie besitzt, um Arbeitsabläufe zu automatisieren [2].

2.1.1 Das prozedurale System

Ein prozedurales System ist nicht zu verwechseln mit Editoren und anderen Tools, die manche Games innerhalb des Spiels oder als separate Anwendung mitbringen. Bei Charakter Editoren (oder Level Editoren), wie man sie aus vielen Rollenspielen kennt, handelt es sich nicht (oder nur in Ausnahmefällen) um prozedurale Systeme.
Die Ergebnisse, die hierbei erzeugt werden, sind eine Reaktion auf die vorangegangene Aktion der Anwender*innen, z.B. das Verschieben eines Objektes zur angegebenen Position. [1].

Im Umkehrschluss bedeutet dies jedoch nicht, dass ein prozedurales System die direkte Einflussnahme ausschließt[6].

Ziel ist es, den Algorithmus für die Generierung eines Levels, Charakters oder Objektes zu nutzen – kurz gesagt zum Gestalten. Das Gestalten unter Anwendung eines Algorithmus unterscheidet sich dabei wesentlich vom Gestalten mit klassischen Methoden[7]. Es muss umgedacht werden.

2.1.1.1 Arbeiten am Algorithmus

Frieder Nake[8] (*1938 in *Stuttgart*), einer der Pioniere der digitalen bzw. generativen Kunst, unterteilt in Ober- und Unterfläche:

> *„Generative Gestaltung zielt auf die Unterfläche und zeigt sich auf der Oberfläche“ [6]*

Mit der Unterfläche ist der Algorithmus gemeint und die Oberfläche steht für die Lösung, also das zu erzeugende Werk. Prozedurale Künstler*innen arbeiten folglich am Algorithmus und gestalten im Zuge dessen lediglich indirekt die Oberfläche.

Um diese Oberfläche zu gestalten, werden dem Algorithmus Regeln auferlegt, die sich aus Bedingungen und Anweisungen zusammensetzen. Alle Regeln, bestehend aus mehreren Schritten, ergeben zusammen ein System, welches in der Lage ist eine Aufgabe automatisiert zu bearbeiten und eine entsprechende Lösung bzw. das gewünschte Ergebnis zu liefern.

Wie dieser Algorithmus mitsamt seinen Regeln tatsächlich aussieht, kann sich je nach Aufgabe und Herangehensweise unterscheiden. Zumeist wird mit einem Algorithmus

---

[6] Verschiedene prozedurale Systeme mit mehr und weniger direkter Einflussnahme bei der Erstellung: https://80.lv/articles/procedural-modeling-for-gamedev/

[7] Mit „Klassischen Methoden“ ist z.B. gemeint: Poly by Poly, Box Modeling oder Sculpting.

[8] Frieder Nake, Database of Digital Art: http://dada.compart-bremen.de/item/agent/68

Programmcode assoziiert – dies muss aber nicht zwingend der Fall sein. Ein solcher Algorithmus lässt sich statt mit dem Schreiben von Programmcode unter anderem mit Nodes[9] umsetzen. Es existieren darüber hinaus prozedurale Systeme, die mit einem eigenen Ansatz arbeiten, wie beispielsweise das *Lindenmayer-System* [7].

Gemeinsam haben alle diese Methoden, sowohl beim Schreiben von Programmcode als auch bei den alternativen Methoden, dass die Handlungsvorschrift in einer Form erhalten bleibt, die sich auch nachträglich umschreiben bzw. bearbeiten lässt, oder um zusätzliche Anweisungen ergänzt werden kann. Somit ist es nicht nur möglich weitere Iterationen am bestehenden Algorithmus vorzunehmen – es lassen sich zudem Ergebnisse beim erneuten Durchlaufen des Algorithmus reproduzieren und bei unterschiedlichem Ausgangspunkt oder angepassten Variablen um neue Ergebnisse ergänzen, da alle vorzunehmenden Schritte in einer lesbaren (bzw. ausführbaren) Form erhalten bleiben [8]. Dies geschieht dabei automatisiert, unter Anwendung der definierten Regeln.

2.1.1.2 Inputs

Je nach Aufgabe ist das Regelwerk jedoch nicht ausreichend. Manche Aufgaben verlangen nach einer Grundlage, auf Basis derer die Regeln angewendet werden.
In einfachen Fällen kann dies eine zufällige Zahl sein, die als Seed[10] dient, wenn die Lösung in keinem weiteren Verhältnis steht. Durch Ändern des Seeds verändert sich die Grundlage, auf die unsere Regeln angewandt werden. Entsprechend verändert sich auch das Ergebnis.

Ist dies nicht der Fall müssen weitere Variablen einbezogen werden, beispielsweise wenn Objekte auf einer Oberfläche verteilt werden sollen. In diesem Fall benötigt der Algorithmus nicht eine Zahl die als Seed dient, sondern ebendiese Oberfläche. Beispielsweise könnte das Regelwerk die Krümmung der Oberfläche auswerten und Objekte entsprechend platzieren.

---

[9] Unter Nodes werden visuelle Programmblöcke verstanden, die Anweisungen enthalten und sich miteinander verbinden lassen.

[10] Als Seed wird eine Zahl bezeichnet, die als Ausgangswert für Berechnungen dient. Durch das ändern der Zahl ergibt sich eine neue Lösung, somit eine Variation.

Bei einem prozeduralen System steht zwischen Aktion des Anwenders und Ergebnis ein Algorithmus, welcher ein Regelwerk anwendet und somit die Basis entsprechend verarbeitet und eine Lösung erzeugt.

> *„[…], a procedural network acts like a recipe, that can be used with different inputs in different shots and different situations with only minor modifications to the inputs and parameters to produce unique results specific to that situation. Assets and tools created this way are said to be live, where the results are easily changeable, based on the inputs. This is in sharp contrast to baked data approaches, where it can be very difficult to make changes." [9]*

2.1.2 Eigenschaften des prozeduralen Workflows

Zusammenfassend lassen sich folgende Eigenschaften eines prozeduralen Systems und dem damit verbundenen Workflows festhalten [10]:

- Arbeitsschritte lassen sich zu jedem Zeitpunkt verändern und ergänzen, ohne dass die darauffolgenden Anweisungen ungültig werden[11].
- Die Handlungsvorschrift ermöglicht später automatisiert zu verfahren
- Bei gleichem Ausgangspunkt lässt sich ein Ergebnis reproduzieren.
- Die Handlungsvorschrift besteht aus mehreren Schritten. Dementsprechend besitzt der Algorithmus eine gewisse Komplexität im Gegensatz zu einfachen Funktionen.
- Es wird ein Input benötigt auf dessen Basis das Regelwerk angewendet werden kann. Dies kann eine Zahl, ein einzelnes Polygon oder komplexere Daten sein.

---

[11] Auch bekannt als „Non-Destructive"-Workflow

2.2 Historie

Wie zum Einstieg erwähnt, sind erste prozedurale Schritte schon im 1980 erschienen *Rogue* vorhanden. Abseits von Games, lassen sich in der Kunst und im Design auch prozedurale Systeme wiederfinden, jedoch oft unter den Begriffen generative Gestaltung oder digitale Kunst. Gemeint sind damit ebenfalls Systeme, die Bilder oder andere künstlerische Arbeiten basierend auf einem Algorithmus generieren.

2.2.1 Digitale Kunst (1960)

Neben *Frieder Nake*[12] (* 1938 in *Stuttgart*) gelten noch *Georg Nees* (* 1926 in *Nürnberg*; † 2016 in *Baiersdorf*) und *A. Michael Noll* (* 1939 in *New York*) als die Pioniere der generativen Kunst. Um 1960 begannen sie erste Programme zu schreiben, die auf Basis eines Regelwerks gestalten. Damals war es eine Ausnahme, Computer zum Generieren von Kunst zu gebrauchen. Da diese vorrangig zum Rechnen dienten und nur wenigen Personen zugänglich waren. Anfangs waren es zumeist noch Linien und Quadrate die zu einer Komposition kombiniert wurden. Das Jahr 1965 gilt als das Geburtsjahr der digitalen Kunst, angefangen mit einer Ausstellung von *Georg Nees* über generative Kunst, in der Studiengalerie der *Technischen Hochschule Stuttgart*.

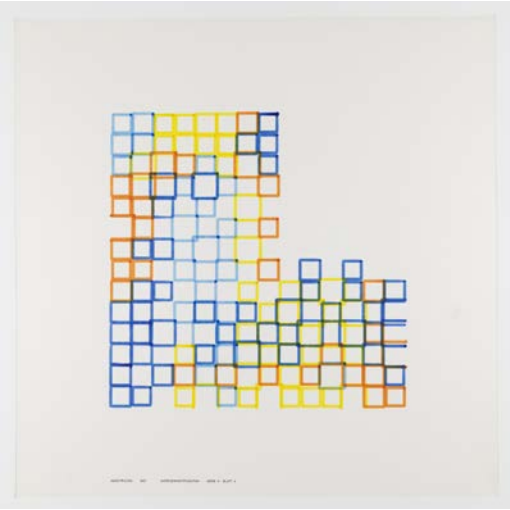

Abbildung 1 Werk von Frieder Nake, 1967 [11]

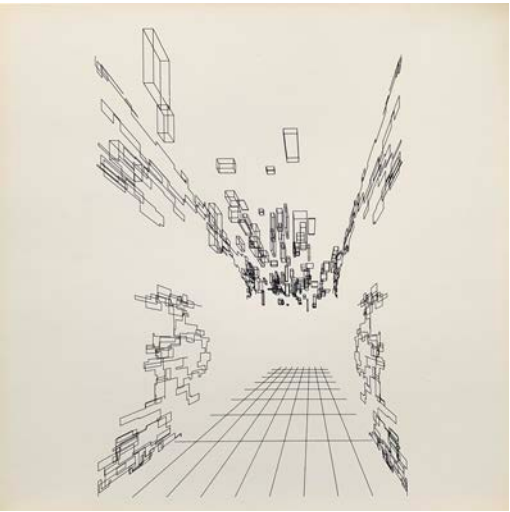

Abbildung 2 Werk von Georg Nees, Hall Corridor, 1970 [12]

[12] Frieder Nake beim Erstellen einer seiner Werke: https://www.youtube.com/watch?v=TV1iol35fHg

2.2.2 Beneath Apple Manor (1978)

Als erstes prozedurales Spiel kann *Beneath Apple Manor*[13] bezeichnet werden, obgleich es weniger bekannt ist und daher diesen Titel an das zwei Jahre später erschienene *Rogue* verloren hat.

Zu Beginn von *Beneath Apple Manor* wird eine Schwierigkeitsstufe und eine Anzahl an Räumen ausgewählt. Daraufhin wird ein neues prozedurales Level, samt Monster und Schätzen, generiert. Das Ziel des Spiels ist es, den goldenen Apfel im letzten Raum zu finden [13].

2.2.3 Demoszene (1980)

Auf Algorithmen basieren außerdem die Kunstwerke der *Demoszene*[14].
Ziel ist es hierbei ein ausführbares Programm zu kreieren, welches in Echtzeit Animationen und Musik darstellt. Das Ergebnis sind meist kreative Kurzfilme, die an Musikvideos erinnern. Das Ganze geschieht unter der Prämisse, möglichst wenig Ressourcen zu verwenden. Die Anwendungen sind meist kleiner als 64 KB. Erst beim Start der Anwendung werden alle visuellen und akustischen Komponenten generiert.

Ihren Ursprung hat die *Demoszene* in den Crack Intros[15] gegen Ende der 1970er Jahre. Um Aufmerksamkeit zu generieren begannen Cracker Gruppen, die den Kopierschutz von Spielen aufhoben, ihr eigenes Intro einzubinden. Um die zunehmend aufwendiger werdenden Intros begann sich ein Kult zu entwickeln, bei dem es nur noch um audiovisuelle Erlebnisse ging.

---

[13] Download zu Beneath Apple Manor: http://worth.bol.ucla.edu/

[14] Sammlung von Beiträgen aus der Demoszene: https://www.heise.de/thema/Demoszene

[15] Bekannte Crack Intros: https://www.youtube.com/watch?v=SFqBkSJOYOQ&t=2609s

2.2.4 Rogue (1980)

Das Spiel *Rogue* ist 1980 erschienen und wird oft als das erste grafische (Grafiken auf Basis von ASCII-Zeichen) prozedurale Spiel bezeichnet. Gleichzeitig hat das Spiel ein eigenes Genre eröffnet: Rogue-like. Dies zeichnet sich neben Permadeath[16] auch durch prozedurale Level aus.

Die prozedurale Generierung ist wahrscheinlich unter anderem der Grund für den großen Erfolg des Spiels. Jede Runde findet in einem einzigartig verzweigten Dungeon statt. Somit ergibt sich immer wieder eine neue Herausforderung für Spieler*innen.

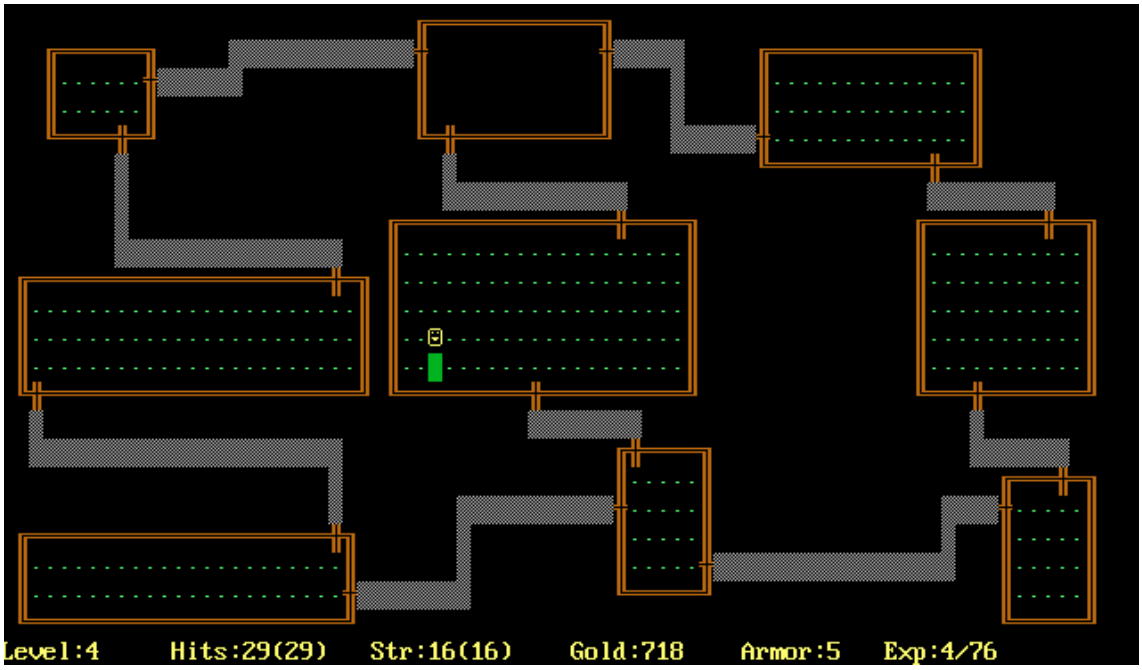

Abbildung 3 Gameplay Szene aus Rogue [14]

[16] Bei Spielen mit Permadeath gibt es nicht mehrere Leben, die es den Spieler*innen erlauben an einer vorherigen Stelle wieder einzusteigen. Ist die Spielfigur gestorben beginnt das Spiel erneut von Anfang an.

2.2.5 Prisms (1987)

Prisms ist eine 3D- und Animationssoftware, mit einem starken prozeduralen Ansatz. Unter anderem sind darin gängige prozedurale Systeme integriert, wie beispielsweise das *Lindenmayer-System* oder der *Perlin Noise*. 1996 wurde die Software von *SideFX* gekauft und wir seitdem unter dem Namen *Houdini* weitergeführt [15]. Über die Jahre hat sich die Software zu einem umfangreichen Paket entwickelt, das weiterhin den Fokus auf prozedurale Workflows legt. Besonders innerhalb der letzten Jahre konnte *Houdini* in der Games-Branche an Einfluss gewinnen.

2.2.6 .kkrieger (2004)

.kkrieger ist ein Shooter, entwickelt von *.theprodukkt*[17] (ehemals *Farbrausch*[18]) aus der Demoszene. Ähnlich wie bei den Kurzfilmen aus der Demoszene, liegt auch *.kkrieger* einem bzw. mehreren Algorithmen zugrunde. Entwickelt wurde es mit der Software *.werkkzeug*. Dabei werden nicht die fertigen Daten gespeichert, sondern der Code wie diese erzeugt werden [16]. Statt der 200-300 MB, die das Spiel mit herkömmlichen Methoden benötigen würden, ist es lediglich 96 KB groß.

2.2.7 Processing (2008)

Im Jahr 2008 erschien die erste Beta-Version der Programmiersprache *Processing* samt zugehöriger Entwicklungsumgebung. Über die Jahre hinweg hat sich *Processing* zu einem der ersten Anlaufpunkte für Künstler*innen entwickelt, die auf Basis von Programmcode prozedurale Grafiken, Animationen und Simulationen verwirklichen wollen [17].

---

[17] .theprodukkt: http://www.theproduct.de/

[18] Farbrausch: http://www.farbrausch.de/

2.2.8 Spore (2008)

Spore[19] ist ein God Game von den Machern der *The Sims* Reihe (*Maxis)*. Ein wesentlicher Bestandteil des Spiels ist das Sammeln von DNA, um das Aussehen seiner Kreatur zu verändern. Dies hat wiederum Auswirkungen auf Fähigkeiten und Stärken im Kampf.

Der Creature Editor[20], in dem die eigene Kreatur von Runde zu Runde verbessert wird, bietet nahezu grenzenlose Möglichkeiten. Der gesamte Körper (Form, Länge, Positionierung und Anzahl der Gliedmaßen), kann frei gestaltet werden.

Während der Laufzeit wird die Kreatur samt Animationen, unter anderem fürs Laufen, Essen und Kämpfen, prozedural generiert [18].

Abbildung 4 Gameplay Szene aus Spore [19]

[19] Spore: http://www.spore.com/

[20] Spore Creature Editor: http://www.spore.com/trial/

2.3 Prozedurale Inhalte

Etwas, das als digitale Kunst anfing, entwickelte sich zu einem motivierenden Element. Dies sorgte dafür, dass viele Spieler*innen mit einem lediglich auf ASCII-Zeichen basierendem Spiel Stunden verbrachten. Gleichzeitig konnten Speicherlimitierungen umgangen werden. Entsprechend bietet die prozedurale Generierung, neben den visuellen Inhalten – wie sie auch im Verlauf dieser Arbeit erzeugt werden – eine ganze Reihe weiterer Anwendungsmöglichkeiten. In Bezug auf Games lässt sich dazu eine Einteilung in vier Gruppen vornehmen [10] [20]:

2.3.1 Game Bits

Zu dieser Gruppe gehören einzelne Komponenten die dem Spiel angehören, beispielsweise Texturen, Sounds, Animationen und Geometrie.

2.3.2 Game Space

Bei Game Spaces handelt es sich um die Spielumgebungen. Dies können Landkarten oder auch Räume sein, in denen sich das Geschehen abspielt.

2.3.3 Game Systems

Bei Game Systems geht es um komplexe Relationen von verschiedenen Elementen, die sich gegenseitig beeinflussen. Beispielsweise ein Wald, der nach einem bestimmten Schema aufgebaut ist. Die verschiedenen Pflanzen folgen Regeln, die festlegen, wie und wo sie wachsen. Aber auch lebensnahe Städte lassen sich hiermit erzeugen [21].

2.3.4 Game Design

Bei der prozeduralen Erzeugung von Game-Designelementen handelt es sich um Spielregeln, Ziele und die Dramaturgie. Somit lassen sich abwechslungsreiche Missionen erzeugen oder Hintergrundgeschichten für Charaktere erstellen.

2.4 Prozedurale Erzeugung

Ebenso vielfältig wie die Elemente, die sich prozedural genieren lassen, sind die mögliche Gründe aus denen Entwickler*innen auf ein solches System zurückzugreifen. Hierzu zählen z.B. große Mengen an benötigten Assets, sehr detaillierte Welten, eine beschränkte Größe des Teams[21], hoher Wiederspielbarkeitswert oder Level Segmente, die sich dem Schwierigkeitsgrad anpassen sollen [1].

Je nach Einsatzzweck gibt es verschiedene Möglichkeiten die Inhalte passend zu genieren, mit den jeweiligen Vor- und Nachteilen der angewandten Methode.

2.4.1 Online Procedural Generation

Unter *online procedural generation* oder auch *runtime procedural generation*, versteht man eine Generierung während der Laufzeit (Englisch: runtime).

Ein typischer Anwendungsfall ist eine unendlich wachsende Spielwelt, innerhalb derer beim Erreichen des Randes immer neue Abschnitte generiert werden. Die Generierung muss jedoch nicht zwingend während des Spielgeschehens erfolgen. Es ist ebenfalls möglich die Ladezeiten zu nutzen, um Charaktere oder Gegenstände prozedural zu generieren, die dann in der darauf folgenden Spielrunde unverändert bleiben [1].

Zwei Prämissen bei dieser Methode sind jedoch zum einen eine schnelle und zum anderen eine qualitative Erzeugung von Ergebnissen [1]. Damit ist unter anderem gemeint, dass die Generierung nicht zu Spielverzögerung führen darf, oder zu massiv längeren Ladezeiten, weil Inhalte erst den Algorithmus durchlaufen müssen. Ein weiterer, jedoch sehr entscheidender Punkt, ist die Lauffähigkeit der Generierungen. Kommt es hierbei zu Problemen, kann dies je nach Fall bedeuten, dass es zur Unspielbarkeit kommt, da beispielsweise Welten generiert werden, die nicht mit dem Gamedesign vereinbar sind.

---

[21] Planet Alpha: https://www.planetalpha-game.com/

Mit wachsenden Möglichkeiten des Algorithmus und damit zunehmender Komplexität, wird somit die Wahrscheinlichkeit größer fehlerbehaftete Inhalte zu generieren [22].

2.4.1.1 No Man Sky

Ein Extrembeispiel für „procedural online generation“, welches in seinem generativen Umfang wahrscheinlich einzigartig ist, ist das 2016 erschienene *No Man Sky* von *Hello Games*. Die Spieler*innen befinden sich in einem Universum, bestehend aus über 18 Trillionen Planeten, mit dem Auftrag zum Mittelpunkt des Universums zu reisen. Jeder Planet auf der Reise ist dabei frei erkundbar und nahezu einzigartig in seiner Beschaffenheit, Vegetation, Klima und seinen Bewohnern.

Als Ausgangspunkt für alle Berechnungen dient dem Algorithmus die Position der Spieler*in im Universum (entsprechend dem so genannten Seed aus Kapitel 2.1). Anhand dieser Information wird die komplette Welt um die Spieler*innen herum bis zum Horizont generiert. Sobald sich die Position dieser verändert, durchläuft die Information erneut den Algorithmus [23].

Zur Generierung der Planetenoberfläche, samt Höhlen und Meeresgrund, werden Fraktale genutzt. Durch Variation der Intensität und Verblendung ineinander ergeben sich unterschiedliche Oberflächen.

Pflanzen, Tiere und andere Details werden aus *Prototypen* generiert. Damit sind Assets gemeint, die ein Artist erstellt. Beispielsweise ist eine Pflanze modular aufgebaut, wodurch sich Elemente ergeben, welche sich mit denen anderer Pflanzen kombinieren lassen. Durch die Kombinationen und Variationen in den Variablen entsteht ein riesiges Repertoire an Assets.

Bei der Generierung kommen zwei Ansätze zur Anwendung, Unterschieden wird dabei zwischen Terrain und Geometrie (z.B. für Tiere, Pflanzen und Felsen).

Abbildung 5 Gameplayszene aus No Man Sky [24]

Bei seinem Release stieß *No Man Sky* jedoch auf starke Kritik. Grund dafür war neben der Story und den Weltraumschlachten vor allem die äußerliche Erscheinung der erzeugten Planeten. Diese waren meist trist, spärlich besiedelt und glichen sich stark untereinander. In den folgenden Jahren veröffentlichte *Hello Games* mehrere große Updates, mit denen der Algorithmus mittlerweile weitaus attraktivere und vielfältigere Planeten erstellen konnte.

2.4.2 Offline Procedural Generation

Infolge dessen ergibt sich auch eine prozedurale Lösung zum Zeitpunkt der Spieleentwicklung: *offline procedural generation*. Bei dieser Methode werden prozedurale Systeme bei der Entwicklung eingesetzt, zur Unterstützung oder zur kompletten Generierung von Inhalten. Oft werden die generierten Lösungen dabei später von einem Artist weiter bearbeitet [25].

Wie bei der online Generierung wird auch bei dieser Methode ein Algorithmus genutzt. Jedoch kann dieser weitaus komplexere Aufgaben bewältigen, da die Ausführung unabhängig von einer laufenden Spielrunde passiert.

Daraus ergeben sich zwei weitere Eigenschaften:

Da die Ergebnisse nach ihrer Generierung statisch sind, können sie auf ihre Qualität geprüft werden, wodurch die Designer*innen die Möglichkeit haben Korrekturen vorzunehmen – beispielsweise durch klassisches Modelling oder Sculpting. Jedoch sind nur Inhalte zur späteren Laufzeit verfügbar, die während der Entwicklung entstanden sind. Dynamische Generierungen sind während einer Spielrunde somit nicht möglich. Zum anderen kann der Algorithmus weitaus komplexere und performancehungrigere Aufgaben bewältigen, da dieser nicht zur Laufzeit ausgeführt werden muss und mehr Systemressourcen für sich beanspruchen darf.

2.4.2.1 Far Cry 5

Das Spiel *Far Cry 5,* welches im Frühjahr 2018 erschienen ist und von *Ubisoft* entwickelt wurde, setzt massiv auf prozedurale Lösungen während der Entwicklung. Wie die Vorgänger, welche ebenfalls von *Ubisoft* entwickelt wurden (mit Ausnahme des ersten Teils), glänzt die Spielereihe mit komplexen, weitläufigen und freierkundbaren Welten.

Abbildung 6 Gameplay Szene aus Far Cry 5 [26]

Der fünfte Teil der Spieleserie handelt von einer radikalen Sekte. Ihr Anführer prophezeit das Ende der Welt und versammelt seine Jünger um sich. Um diesem Treiben ein

Ende zu setzen schlüpfen die Spieler*innen in die Rolle eines namenlosen Deputy, um infolge dessen alle vier Anführer zu beseitigen. Die Handlung spielt in einer fiktiven Region im amerikanischen Bundesstaat Montana. Dementsprechend ist das Erscheinungsbild von weitläufigen Wiesenlandschaften, dichten Wäldern und kleineren Siedlungen geprägt. Um diese weitläufige Welt mit Leben zu füllen, hat das Entwicklerteam mehrere offline arbeitende prozedurale Tools entwickelt.
Als Input dient für die meisten Tools eine *Heightmap*[22]. Anhand des entstehenden Gefälles können ohne Zutun der Designer*innen Klippen extrahiert und automatisiert gestaltet werden.

Für Wälder und Wiesen stehen den Designer*innen verschiedene *Footprints* zur Verfügung. Diese beinhalten Bodengrund, Gräser, Büsche, bis hin zu Bäumen, die jeweils nach realistischem Vorbild zueinander verteilt werden. Mit dem Zeichnen einer Maske auf dem Terrain werden somit Regionen erzeugt und mit Inhalt aus dem jeweiligen *Footprint* gefüllt. Auch existieren Tools bei denen der Artist mehr Kontrolle behält und die ihm die Arbeit lediglich erleichtern.

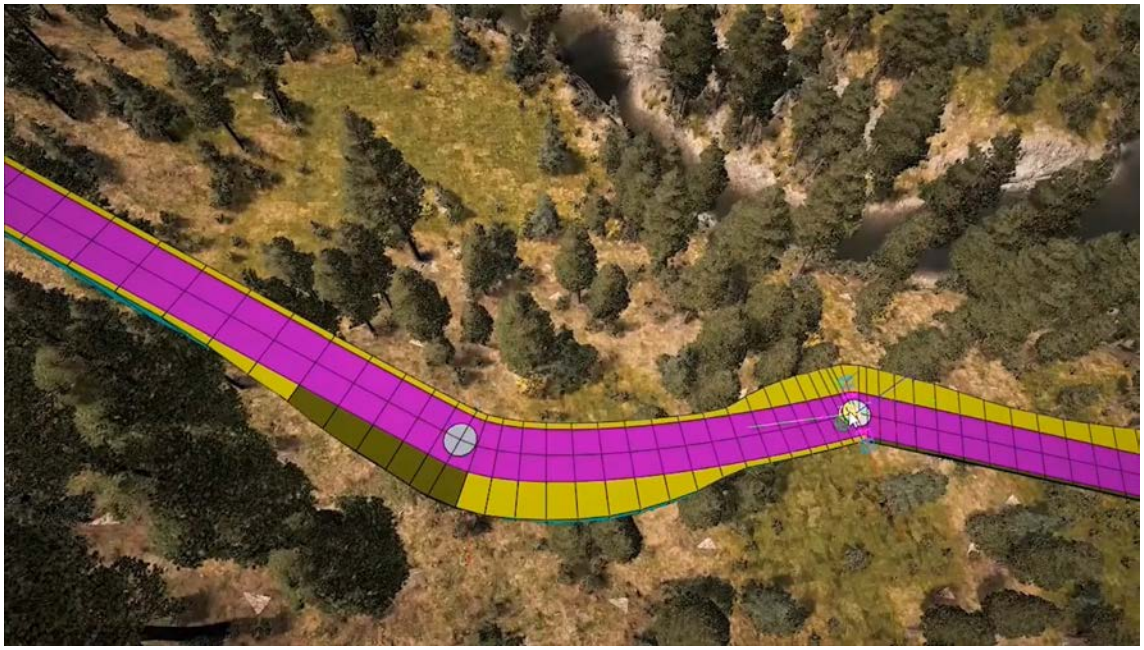

Abbildung 7 Prozedurales Erstellen von Straßen in Far Cry 5, mittels der Anbindung an Houdini [27]

[22] Unter einer Heightmap versteht man eine Höhenkarte. Diese ist eine Textur mit verschiedenen Grauwerten. Hellere Bereiche werden dabei als Berge interpretiert, dunklere als Täler.

Zum einen ein prozedurales Werkzeug zum Erstellen von Straßen, zum anderen eines zum Erstellen von Hochspannungsleitungen. Beide funktionieren nach einem ähnlichen Prinzip. Der Artist zeichnet eine Kurve auf dem Terrain, die anschließend umgewandelt wird – entweder in eine Straßenstrecke mit Fahrbahnbegrenzung oder in Hochspannungsmasten, die durch Hochspannungsleitungen untereinander verbunden sind.

2.4.3 Mixed Procedural Generation

Unter *mixed procedural generation* wird eine Kombination beider Methoden verstanden. Zumeist handelt es sich dabei um im Vorhinein klassisch designte Gegenstände oder Level Segmente, die durch den Algorithmus entsprechend platziert oder kombiniert werden.

Durch diese Vorgehensweise ist zum einen eine Lauffähigkeit der Generierung leichter zu gewährleisten, da der Algorithmus sich beispielsweise auf das Kombinieren von Assets beschränkt, zum anderen bleibt die Möglichkeit dynamische oder wachsende Inhalte zu generieren erhalten.

2.4.3.1 Marbloid

Ein Beispiel dafür ist das Ende 2018 erschienene Mobile Game *Marbloid* von *Supyrb*[23]. Die Spieler*innen steuern eine Kugel durch eine fantastische Welt im Stil des Vaporwave, vorbei an verschiedenen Plattformen, Hindernissen und Schanzen.

Die einzelnen Segmente wurden zuvor offline designt und in Kategorien hinsichtlich des Stils und der Schwierigkeit unterteilt. Durch diesen Ansatz haben die Designer*innen direkten Einfluss auf das visuelle Erlebnis und können unspielbaren Passagen vorbeugen. Im laufenden Spiel sucht der Algorithmus passenden Level Segmente aus und kombiniert sie hintereinander – so ergibt sich in der Theorie ein nie endendes Level.

---

[23] Supyrb: http://www.supyrb.com/

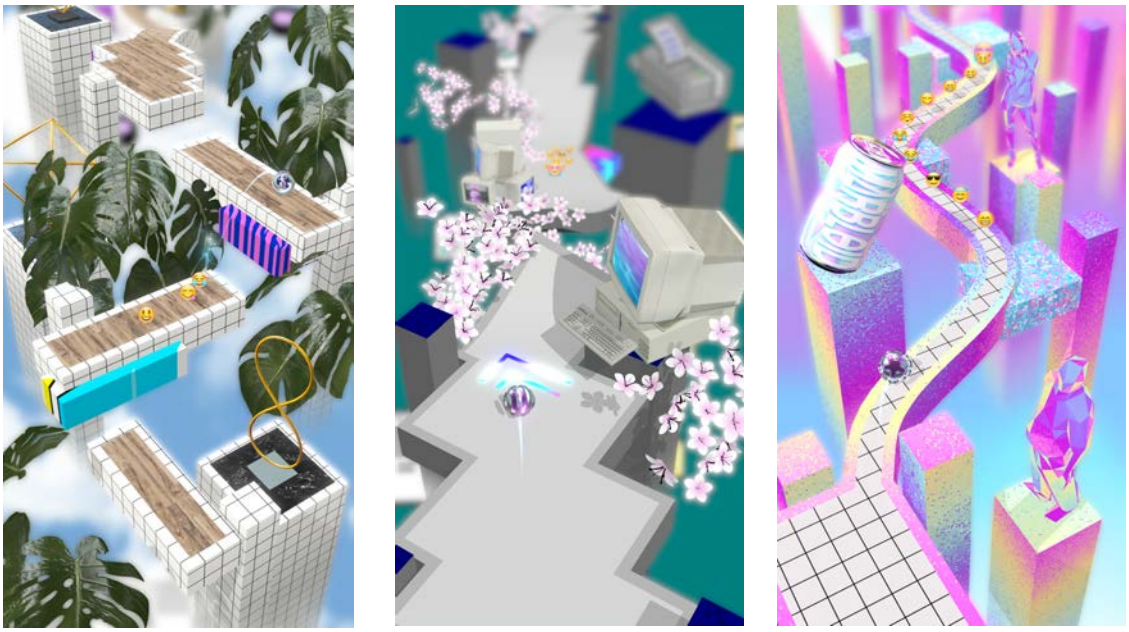

Abbildung 8 Gameplay Szenen aus Marbloid [28]

Die Entscheidung welche Segmente in der aktuellen Spielrunde geladen werden, ist abhängig vom aktuellen Missionsziel und dem allgemeinen Fortschritt im Spiel. Ist es beispielsweise das Ziel eine gewisse Anzahl an Portalen zu durchqueren und der Spieler oder die Spielerin befindet sich noch am Anfang, werden entsprechend häufiger Portale und leichtere Level Segmente platziert.

3 Projekt

In dieser Arbeit soll ein prozedurales System entwickelt werden, welches aus mehreren Komponenten für unterschiedliche Aufgaben besteht und ein Level auf Grundlage simpler Geometrie erstellen kann. Das System soll möglichst selbstständig, ohne zwingend notwendiges Eingreifen der Designer*innen die Aufgabe lösen und ein produktionstaugliches Ergebnis liefern.

Die Gestaltung der Level erfolgt somit ausschließlich durch die Geometrie, die als Input dient und anschließend vom System weiter ausgebaut und verfeinert wird. Die Generierung orientiert sich dabei soweit wie möglich am Input und setzt die Idee der Leveldesigner*innen fort.

3.1 Theoretische Anwendungsfälle

Mögliche Anwendungsfälle für ein solches System könnten unter anderem die Weiterverarbeitung eines Blockings oder die Produktion eines Remakes sein.

3.1.1 Blocking

Unter Blocking, auch als White Box bezeichnet, versteht man eine erste Skizze des Levels. Dieser Entwurf besteht zumeist aus Boxen oder stark vereinfachter Geometrie, die einen ersten Eindruck vermitteln soll und als Platzhalter für spätere Assets dient.

Statt das Blocking ausschließlich als Platzhalter zu nutzen, kann es zu einem Gestaltungswerkzeug werden, welches in die Level-Erstellung einfließt, um im Nachhinein daraus detaillierte Geometrie zu erzeugen [29].

3.1.2 Remake

In den letzten Jahren entwickelte sich der Trend alte Spieleklassiker als Remastered Titel erneut auf den Markt zu bringen. Bei einer Remastered Version werden Texturen oder Meshes gegen höher aufgelöste Varianten ausgetauscht.

Bei einem Remake hingegen entsteht, zumindest technisch gesehen, ein neues Spiel, welches sich an seinem Vorgänger orientiert. Diese Orientierung kann mehr oder weniger stark ausfallen. Für ein authentische, zeitgemäße Neuauflage des Vorgängers ist es möglich die vorhandene Geometrie aufzuarbeiten oder komplett neu zu erstellen. Angefangen von simplen Assets bis hin zu gesamten Leveln.

3.2 Tomb Raider

In dieser Arbeit soll ein Level prozedurales generiert werden. Als Beispiel dient das Originallevel eines Spieleklassikers: *Tomb Raider* (Teil 1) von *Eidos*, aus dem Jahr 1996. Der Titel entstand zum Anfang der Ära der 3D-Spiele. Entsprechend ist die Technik noch in den Kinderschuhen Die Texturen sind grob aufgelöst und die Menge an Polygonen eines gesamten Levels entspricht heutzutage denen eines einzelnen Assets.

Aufgrund der großen Modding-Community[24] stehen zahlreiche Tools von Fans[25] und den Entwicklern selbst zur Verfügung. Diese Tools ermöglichen es die original Geometrie samt Texturen aus den ersten *Tomb Raider* zu extrahieren und als Basis für das prozedurale System zu verwenden.

3.2.1 Tomb Raider – Beginn des ersten Levels

Nach der Intro-Sequenz verschließen sich die Tore und der einzige Weg führt tiefer in die Höhle hinein. Diese ist zu Anfang mit Schnee bedeckt, mit dem weiteren Eintauchen in die Höhle werden die Wege allerdings verzweigter und der Schnee nimmt ab. Nun

[24] Tomb Raider Community https://www.tombraiderforums.com/

[25] Tomb Raider Map Extractor https://www.tombraiderforums.com/showthread.php?t=219901

sind die Flächen nur noch sporadisch von Schnee oder Eis bedeckt. Je weiter der Spieler oder die Spielerin in die Höhle gelangt, umso grüner wird die Umgebung und die umliegenden Höhlenwände werden stellenweise von grobem Mauerwerk verziert. Dieser erste Abschnitt dient als dramaturgischer Übergang von einer eingeschneiten Höhle zu einer verlassenen Stätte in den Tiefen des Berges mit tropisch anmutendem Klima.

Abbildung 9 Bilderreihe aus dem ersten Abschnitt des Levels aus Tomb Raider [30]

3.2.2 Anforderungen an das prozedurale System

Das Originallevel stellt eine Reihe von Anforderungen an das prozedurale System, welche durch verschiedene Komponenten gelöst und am Ende zu einem Ganzen zusammengefügt werden.

Angefangen bei der grundlegenden Levelgestaltung: Die Spieler*innen treffen auf Abschnitte in der Höhle, die ausschließlich durch Springen oder Klettern erreicht werden

können. Daher müssen Abstände eingehalten werden, um nicht unüberwindbare Klüften zu generieren. Gleichzeitig besitzen manche Passagen eine sehr geringe Deckenhöhe, was dazu führen kann, dass diese Abschnitte unpassierbar werden.
Somit muss die Illusion von Felsstruktur geschaffen werden, ohne wesentlich von den grundlegenden Proportionen der Höhle abzuweichen.

Neben der Felsstruktur gibt es auch Flächen, die aus Mauerwerk bestehen. Statt einer einfachen Textur sollen plastisch wirkende Ziegel verteilt werden, die für eine vielfältige Oberflächenstruktur sorgen.

Nicht zuletzt beheimatet das Höhlensystem verschiedene Klimazonen. Angefangen bei Eis und Schnee, über gemäßigte Zonen, bis hin zu Wänden welche durchgehend mit Pflanzen bedeckt sind. Dieses Zusammenspiel soll im prozeduralen System durch dicht miteinander verbundene Komponenten abgebildet werden.

3.3 Software

Zur Erstellung des hier dokumentierten prozeduralen Systems wird *Houdini* in Kombination mit der *Unreal Engine verwenden.*

Wie zu Beginn dieser Arbeit beschrieben[26], gibt es verschiedene Methoden, um ein prozedurales System zu erstellen. Eine der bekanntesten Softwarelösungen ist *Prisms*[27], welches mittlerweile von *SideFX* unter dem Namen *Houdini* vertrieben wird.

Houdini findet bereits seit langem Einsatz als VFX-Software und ist in diesem Bereich ein häufig genutztes Tool. In den letzten Jahren gelang ihm vermehrt auch der Einzug in die Spielebranche[28] [31]. Mittlerweile ist *Houdini* die meistgenutzte Software, um prozedurale Lösungen für Spiele zu entwickeln – nicht zuletzt auf Grund ihrer hervorragenden Integration in die gängigsten Game Engines[29] [9].Prisms (1987)

---

[26] Vgl. Kapitel: Arbeiten am Algorithmus, 2.1.1.1
[27] Vgl. Kapitel: Prisms (1987), 2.2.5
[28] Vgl. Kapitel: Far Cry 5, 2.4.2.1
[29] Vgl. Kapitel: Prisms (1987) 2.2.5

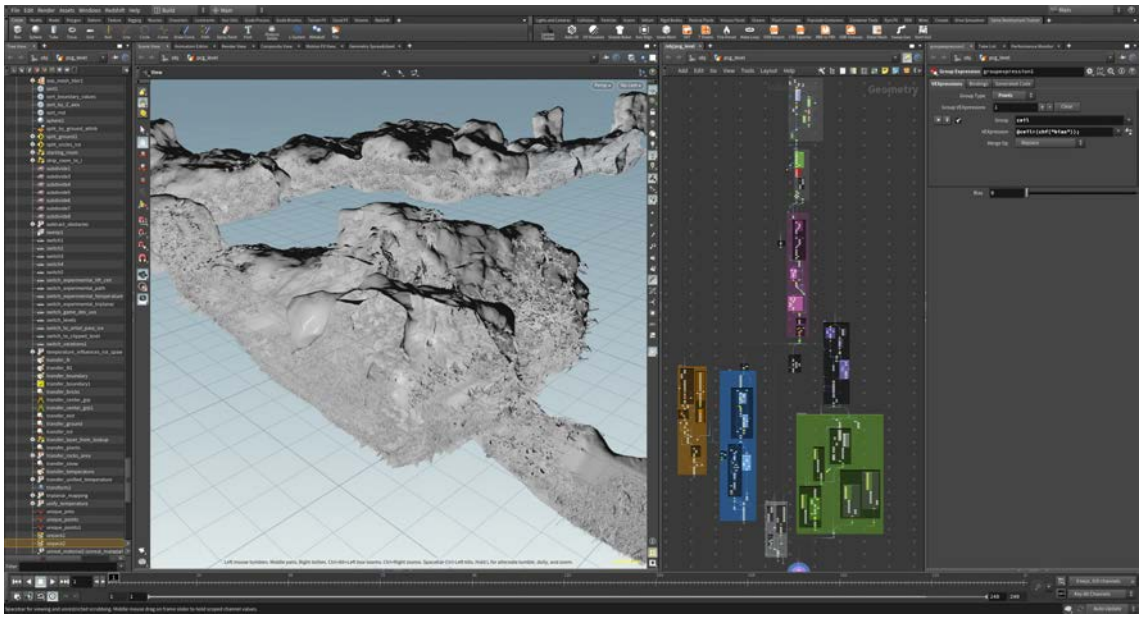

Abbildung 10 SideFx – Houdini (Version 17.5.229) Interface

Hierzu zählt auch die *Unreal Engine* von *Epic Games*, die im Zuge dieser Arbeit verwendet wurde. Durch die Anbindung über die *Houdini Engine* werden Funktionen wie das automatische Erstellen von Instanzen ermöglicht.

4 Dokumentation

Die folgende Übersicht der Dokumentation, sowie die darauffolgende detaillierte Beschreibung der Grundprinzipien der Komponenten, orientiert sich an der Gliederung und Reihenfolge des Node Graphs in *Houdini* [30].

Zu Beginn des Systems steht der Import des Originallevels[31] und die Erzeugung aller benötigten Attribute. Die Geometrie wird aufbereitet und zur Deformation temporär in ein Volumen umgewandelt[32]. Dieses wird schrittweise geformt bis die gewünschte Struktur der Höhle erreicht ist. Nach der abschließenden Deformation werden die Volumen zurück zu einem Mesh konvertiert. Dieses dient als Ausgangspunkt für die weiteren Komponenten des prozeduralen Systems.

Zur Erzeugung der Eiszapfen[33] werden bereits bestehende Flächen an der Höhlendecke verwendet. Auf diesen Flächen werden Punkte verteilt, an denen potenziell Eiszapfen wachsen können.

Für die Ziegel[34] wird eine dreidimensionale Struktur generiert, die flach im Raum platziert wird. Um dieses Muster an seine endgültige Position zu bringen erfolgt eine Rückprojektion anhand der UV-Koordinaten.

Pflanzen und Steine[35] werden anhand von Oberflächenwerten platziert. Die Geometrie erhält einen Ausgangswert der durch verschiedene Faktoren, z.B. Temperatur, verändert wird. Dieser neue Wert steuert die Dichte und Art der Verteilung.

---

[30] Vgl. Abbildung 11 Der gesamte Node Graph des prozeduralen Systems in Houdini
[31] Vgl. Kapitel: Import, 4.1
[32] Vgl. Kapitel: Umwandlung in Volumen, 4.2.2
[33] Vgl. Kapitel: Eiszapfen, 4.4
[34] Vgl. Kapitel: Ziegel, 4.6
[35] Vgl. Kapitel: Vegetation, 4.5

Als letzter Schritt in *Houdini* wird die Höhle, welche in den vorherigen Modulen den Ausganspunkt gebildet hat, von allen weiteren Attributen befreit und mit Decals versehen[36]. Die gesammelte Geometrie wird anschließend in *Unreal* mit einem Material[37] versehen, welches in der Lage ist verschiedene Klimazonen abzubilden.

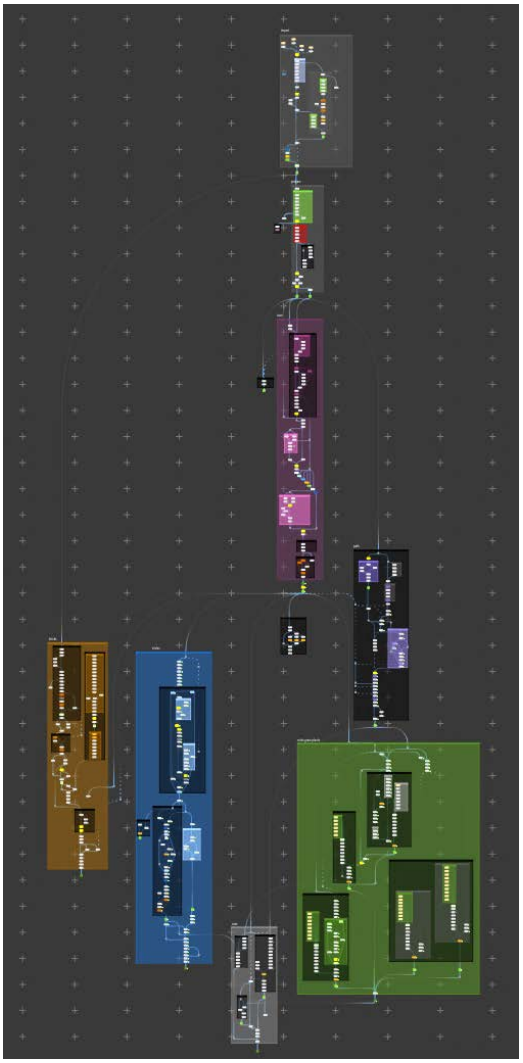

Abbildung 11 Der gesamte Node Graph des prozeduralen Systems in Houdini

Import des Originallevels (erste Gruppe oben, grau), Erzeugen von zusätzlichen Attributen (grau), Generierung der neuen Höhle und Übertragung von Attributen (lila), Pfaden (schwarz), Ziegeln (braun/orange), Eis (blau), Höhle Aufbereitet für den Export (helles Grau) und Erstellung von Vegetation (grün).

[36] Vgl. Kapitel: Höhle, 4.7

[37] Vgl. Kapitel: Materials, 4.8

4.1 Import

Mit dem Importieren der Level erhält man zusätzliche Daten, die innerhalb der Geometrie erfasst wurden. Neben den allgemeinen Positions- und UV-Koordinaten weist die importierte Geometrie eine Besonderheit auf: Level-Abschnitte wurden vorab durch Raumnummern gruppiert.

Insgesamt stehen folgende Daten (zusätzlich zu den allgemeinen Geometriedaten) zur Verfügung:

Typ	**Beschreibung**
N	Normalen
UV	UV- Koordinaten
Group: Rooms	Gruppierung der Polygone anhand von Raumnummern

4.1.1 Typisierung

Eine relevante Information, die nicht in den Geometriedaten enthalten ist, ist der Typ oder die Kategorie des Inhaltes, die das Polygon darstellt. Im Level gibt es unterschiedliche Untergründe und Objekte, die nur durch die Textur angedeutet werden. Diese Informationen gilt es innerhalb der Geometrie für die anschließende Generierung festzuhalten. Wird der dazugehörige Textur Atlas gegen ein Checkerboard[38] ausgetauscht, lässt sich ein Muster bei der Texturierung erkennen.

Jede UV-Koordinate kann einem exakten Bereich auf dem Texture Atlas zugeordnet werden. Wird nun eine Liste erstellt mit dem Typ, den der jeweilige Texturblock abbildet, kann anhand der UV-Koordinaten die Typisierung des Polygons erfolgen.

[38] Unter einem Checkerboard versteht man eine Textur, die an ein Schachbrett erinnert. Diese wir zur Überprüfung des Texture-Mapping verwendet und gibt Rückschluss auf die UV-Koordinaten.

Abbildung 12 Texture Atlas der ersten Level

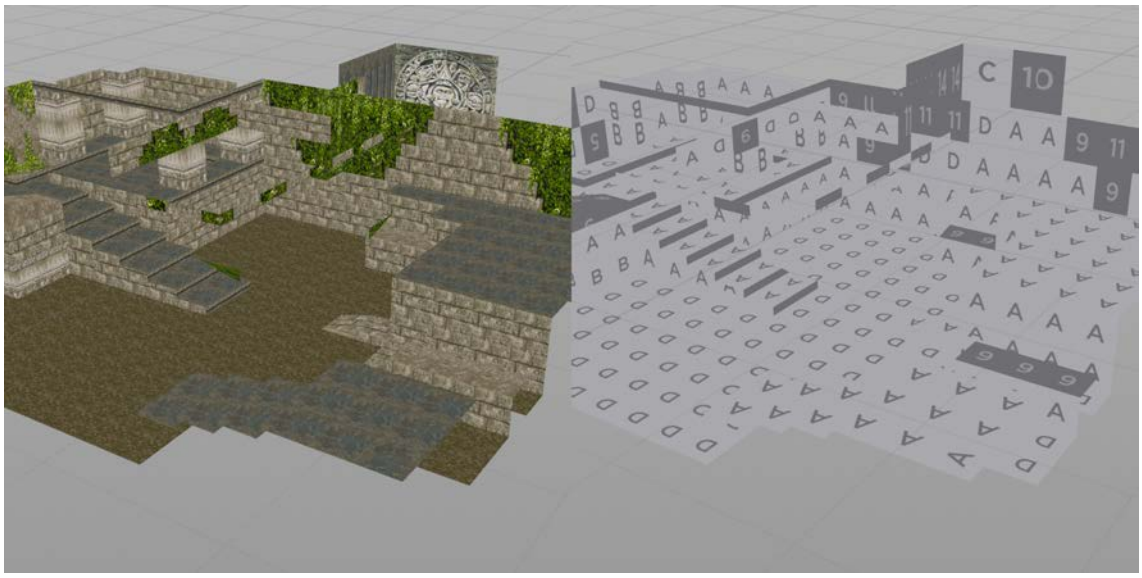

Abbildung 13 Gegenüberstellung der texturierten Level und Level mit Checkerboard, zur Visualisierung der UV-Koordinaten

Das Konzept ähnelt einem Look Up Table[39]: Hier werden vorhandene Daten auf Basis neuer Informationen umgewandelt. Somit kann der Typ eines Polygons anhand des gesetzten Attributs bestimmt werden und nicht mehr ausschließlich visuell. Darüber hinaus wurden die UV-Koordinaten, welche denselben Texturblock abbilden, durch eine gemeinsame Layer-Nummer gekennzeichnet. Jeder der möglichen Typen (*Bricks*, *Snow*, *Plants*, *Ice*) besitzt eine Liste, in der die Layer-Nummer eingetragen wird und womit der Typ auf das Polygon geschrieben werden kann.

[39] Look Up Tables werden unter anderem beim Grading verwendet um Farben neu zuzuordnen.

4.1.2 Temperature

Zusätzlich wird das Aussehen der Höhle stark durch die unterschiedlichen Klimazonen geprägt. Die Temperatur variiert von Eis hin zu neutralen bis fast tropischen Abschnitten. Direkte Informationen zur Temperatur sind der Geometrie nicht angehangen, jedoch lässt sich auf Basis der zuvor erstellten Typen ein ungefähres Bild konstruieren.

Der Ausgangswert für alle Polygone ist zunächst 0,5. Den Flächen, die Pflanzen abbilden, wird ein positiver Wert zugeteilt (+0,5). Eis und Schnee hingegen erhalten einen negativen Wert (-0,5).

Damit die Temperaturübergänge fließend sind werden die Werte in mehreren Iterationen weichgezeichnet. Das Ergebnis sind sich langsam ausbreitende Zonen mit unterschiedlicher Temperatur.

Die Funktion des ersten Levelabschnittes ist es unter anderem einen visuellen Übergang von der außerhalb der Höhle herrschenden Kälte, bis hin zu tropischen Temperaturen zu schaffen. Entsprechend wird eine zusätzliche Möglichkeit benötigt, um die Temperaturen über das gesamte Level hinweg zu regulieren.

Dazu wird anhand der relativen Position der Polygone zur Bounding Box[40] ein Gradient erstellt. Dieser kann nach Belieben transformiert werden, um die Ausrichtung des Temperaturverlaufs zu steuern. Um eine kontinuierliche Veränderung der Temperatur zu erhalten, unter Beibehaltung der originalen Temperaturverteilung, werden beide Varianten miteinander kombiniert. Zusätzlich besteht die Möglichkeit zwischen den beiden Lösungen hin und her zu blenden. Somit können fehlerhafte und nicht sinnvolle Ergebnisse umgangen werden.

[40] Eine Bounding Box umgibt die gesamte Geometrie und definiert somit die Größe und Proportionen.

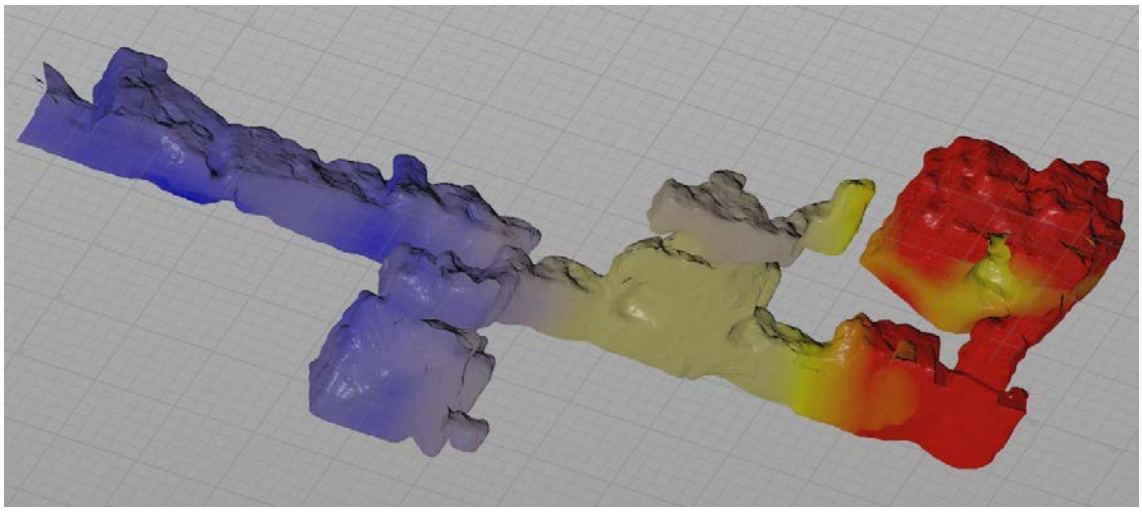

Abbildung 14 Visualisierung des Temperatur Attributs, vom Anfang mit Schnee bedeckten des Levels (blau) bis hin zum tropischen Abschnitt (rot)

4.1.3 Ground

Anhand der Oberflächennormalen wird die Ausrichtung eines Polygons im Raum beschrieben. Unter Zuhilfenahme eines zweiten Vektors kann auch die Ausrichtung zu diesem bestimmt werden.

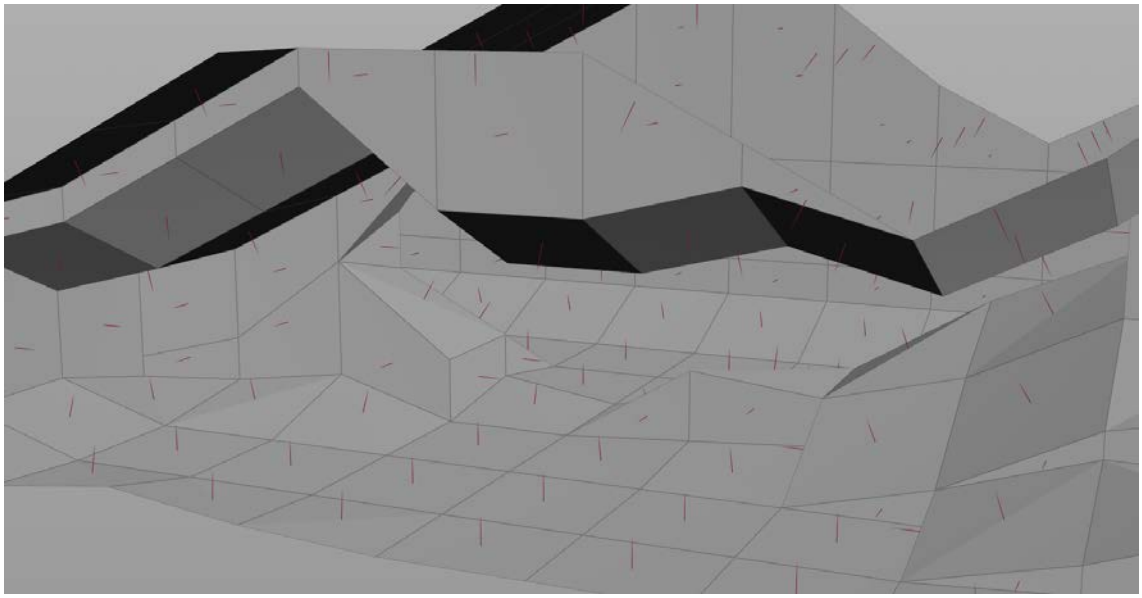

Abbildung 15 Die Ausrichtung der Oberflächennormalen, visualisiert durch Pfeile

Somit lassen sich alle Polygone gruppieren, die nach oben in Richtung der positiven Y-Achse ausgerichtet sind. Diese Gruppe (in Folge *Ground* genannt) kennzeichnet den Boden, auf dem die Spieler*innen sich bewegen können. Dieses Attribut ist von Bedeutung, um den Untergrund vor zu starken Veränderungen zu bewahren, wodurch das Gameplay eingeschränkt werden könnte – oder auch um Gräser, Pflanzen und Steine korrekt zu platzieren.

4.2 Aufbereitung

Ein erster Ansatz sah vor, die auf der Geometrie basierten Daten zu belassen und das Level durch unterschiedliche Deformationen der Polygone von der ursprünglichen, blockigen Form zu befreien. Ein Vorteil dieser Methode ist, dass sie eine sehr präzise Deformation erlaubt. Die bestehenden Punkte werden gezielt durch einen Vektor in eine neue Position gebracht.

Hierbei kommt es jedoch bei einer übermäßig starken Deformationen der Grundstruktur zu einer Überlagerungen der Polygone. Um dieses Problem zu umgehen werden in dieser Arbeit zwei unterschiedliche Ansätze kombiniert.

4.2.1 Übergänge

Das Originallevel verfügt über harte Übergänge zwischen Boden und Wänden – die Flächen treffen orthogonal aufeinander. Das Ziel ist es, diese orthogonal aufeinandertreffenden Flächen zu identifizieren und entsprechend zu verschieben, um einen natürlicheren Übergang zu erzeugen.

Werden die äußeren Punkte der Gruppe *Ground*[41] betrachtet, die keine weiteren Nachbarn aufweisen, ergeben sich daraus die Punkte, die das Ende bzw. den Anfang von Boden und Wand definieren.

---

[41] Vgl. Kapitel:

Durch die Abtrennung der Polygone entstehen an diesen Stellen sich überlagernde Punkte, mit den Normalen für Boden bzw. Wand. Werden die Normalen dieser sich überlagernden Punkte verglichen[42], so kann die Ausrichtung der Flächen zueinander bestimmt werden. Sollte die Ausrichtung zueinander orthogonal sein, werden die Punkte in Richtung der positiven Y-Achse verschoben, womit sich eine leichte Steigung ergibt.

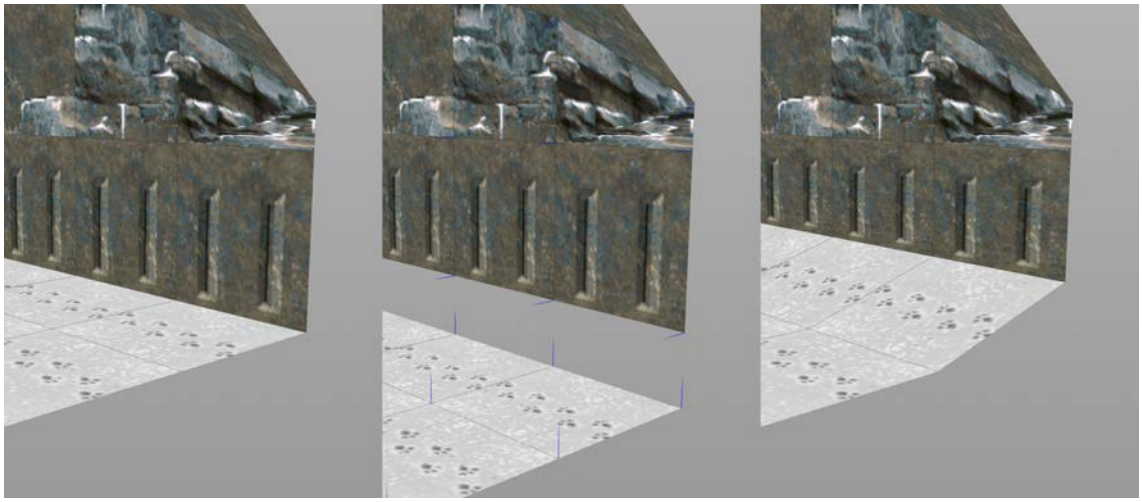

Abbildung 16 Ursprünglicher Winkel zwischen Boden und Wand (Links); Boden und Wand voneinander getrennt und die Normalen durch blaue Pfeile visualisiert (Mitte); Neuer Übergang zwischen Boden und Wand (Rechts)

4.2.2 Umwandlung in Volumen

Wie eingangs erwähnt eignet sich eine Deformation der Polygone nur bedingt um die endgültige Höhlenstruktur zu erhalten. Abhilfe schafft die temporäre Umwandlung in ein Volumen während der Deformation. Bei einem klassischen Volumen kann es jedoch zu einer fehlerbehafteten Darstellung beim Zurückwandeln in ein Mesh kommen.

Ground, 0

[42] Das Skalarprodukt gibt Auskunft über den Winkel zweier Vektoren zueinander: https://www.sidefx.com/docs/houdini/nodes/vop/dot.html

4.2.2.1 SDF – Signed Distance Field

Bekannte Probleme sind Treppeneffekt (ähnlich wie man sie von Pixelgrafiken kennt) oder andere unschöne Artefakte. Besser geeignet ist hier ein Signed Distance Field[43] (auch Signed Distance Function genannt) – künftig mit SDF abgekürzt.

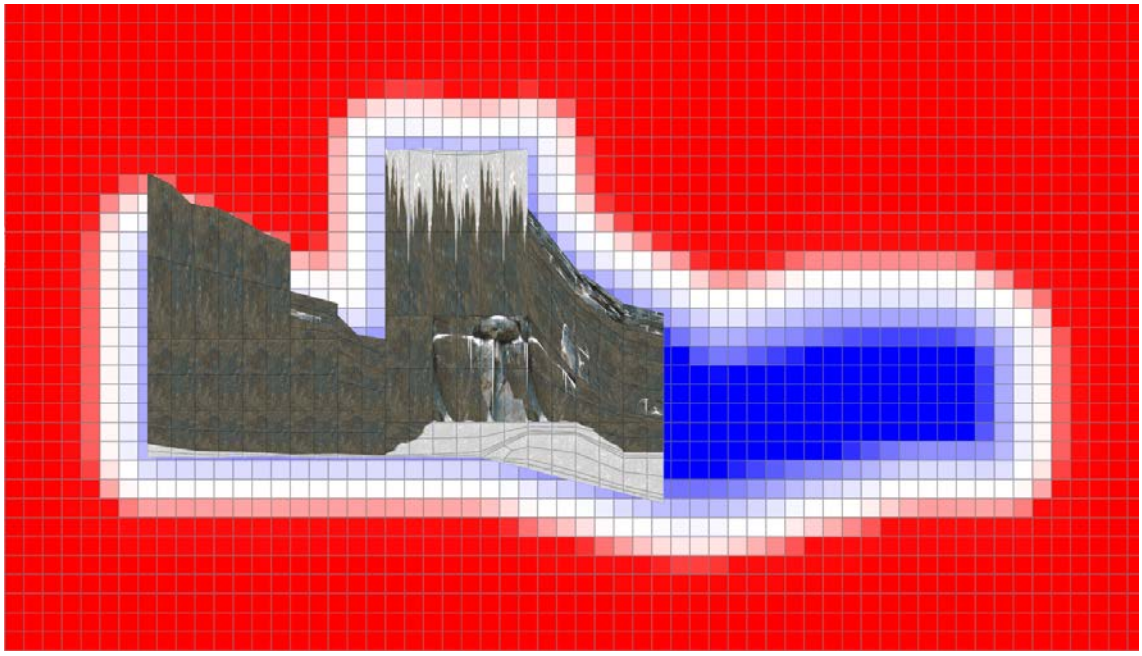

Abbildung 17 Visualisierung der Voxels des SDFs bei eingeblendetem Level. Rote Flächen besitzen einen positiven Wert, blaue Fläche einen negativen. Je heller die Fläche, desto größer die Annäherung an 0

Ein SDF ist in der Lage geschwungene Oberflächen präziser darzustellen. Dazu enthalten die Voxels eine Angabe über die Entfernung zur nächstgelegenen Oberfläche. Einen positiven Wert erhalten diese wenn sich der Voxel außerhalb des Objektes befindet. Einen negativen Wert erhalten sie, falls dieser sich innerhalb des Objekts befinden.
Je größer der Wert, desto weiter entfernt befindet sie der jeweilige Voxel von der Oberfläche.

[43] Zur Konvertierung der Polygone in ein SDF wird die *VDB from Polygons* Node genutzt. Mehr Informationen zu Volumes und SDFs in Houdini: http://www.tokeru.com/cgwiki/index.php?title=HoudiniVolumes

4.2.2.2 Gradient

Aus dem SDF lässt sich wiederum ein weiteres Volumen erzeugen, ein Gradient. Beim Gradienten handelt es sich um ein Volumen welches mit den Oberflächennormalen vergleichbar ist und somit Richtungsvektoren für das SDF bildet.

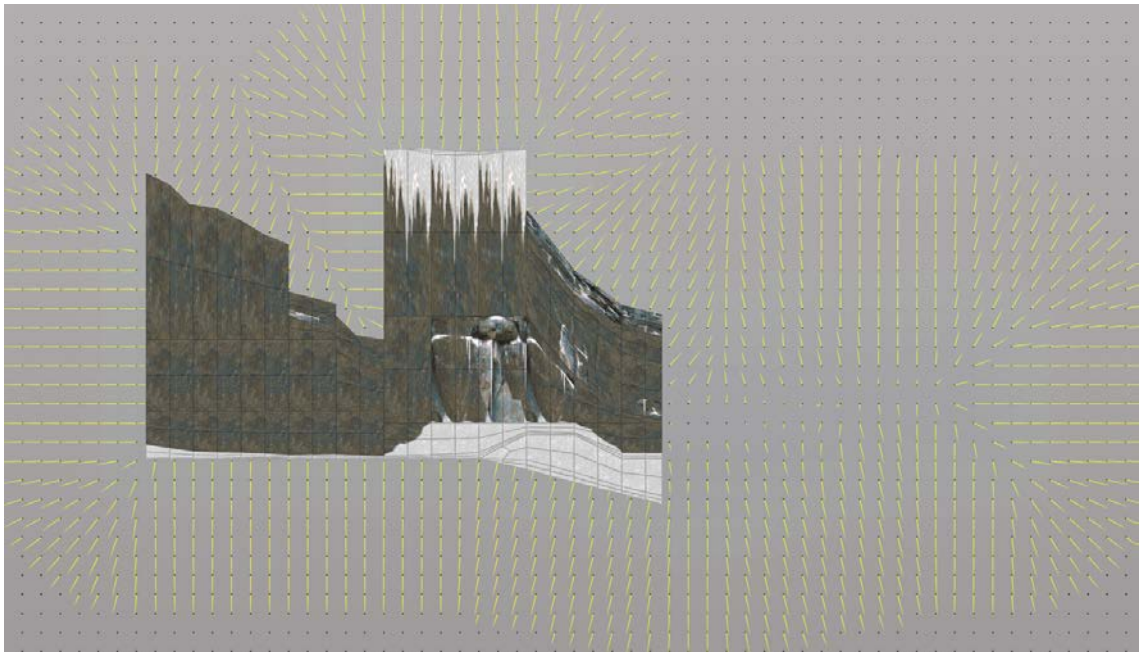

Abbildung 18 Visualisierung der Vektoren des Gradienten bei eingeblendetem Level

Somit ergeben sich eine Reihe von verschiedenen Volumen:

- SDF (Typ Float) zur Abbildung der Oberfläche
- Gradient (Typ Vektor) als Oberflächenvektor
- Klassische Volumen (Typ Float) für Attribute wie *Ground*, Snow, *Plants* und *Bricks*

Durch die Multiplikation eines Noises (Rauschfunktion) mit dem Gradienten, werden die Richtungsvektoren verstärkt, gedämpft oder umgekehrt. Das Volumen *Ground*, in dem die Voxels Informationen über den Untergrund enthalten, wird ebenfalls in die Kalkulation der Richtungsvektoren einbezogen. Diese Voxels sollen einen dämpfenden Einfluss auf die Richtungsvektoren haben und somit den Boden vor zu starker Deformation schützen.

Der bearbeitete Gradient wird nun genutzt, um die anderen Volumen (SDF und Attribute) anhand ihrer Richtungsvektoren zu deformieren.

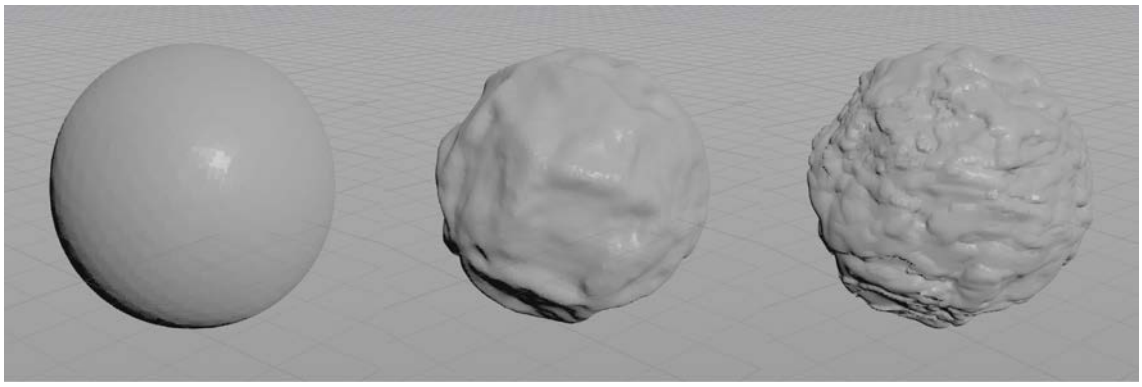

Abbildung 19 Deformation des SDFs in mehreren Schritten bei gleichzeitiger Reduktion der Voxelgröße am Beispiel einer Kugel

Die Deformation der Volumen erfolgt in mehreren Schritten. In den ersten sind die Deformationen deutlich sichtbar und die Grundstruktur wird bestimmt. In den darauffolgenden Schritten wird die Voxelgröße reduziert und der Einfluss des Gradienten wird zunehmenden verringert. Das Ergebnis sind immer feiner werdende Strukturen.

Ist die letzte Detailstufe erreicht, wird das SDF wieder in ein Mesh umgewandelt und alle Volumen projizieren ihre Attribute auf das entstandene Mesh.

4.2.2.3 Projektion auf reduzierte Geometrie

Ein Nachteil bei der Umwandlung in ein Volumen ist, dass eine sehr kleine Voxelgröße benötigt wird, um scharfe Strukturen und Kanten abzubilden. Eine so stark reduzierte Voxelgröße in Verbindung mit weitläufiger Geometrie bringt selbst aktuelle Hardware an ihre Grenzen.

Um lange Kalkulationen zu umgehen wird eine Kopie des Levels erzeugt und diese anschließend auf ein paar Polygone reduziert. Sporadisch werden nun kleinere Bereiche des hochaufgelösten Levels auf die reduzierte Kopie projiziert, wodurch Details wie plane Flächen und gradlinige Kanten vom Low-Poly Model übertragen werden.

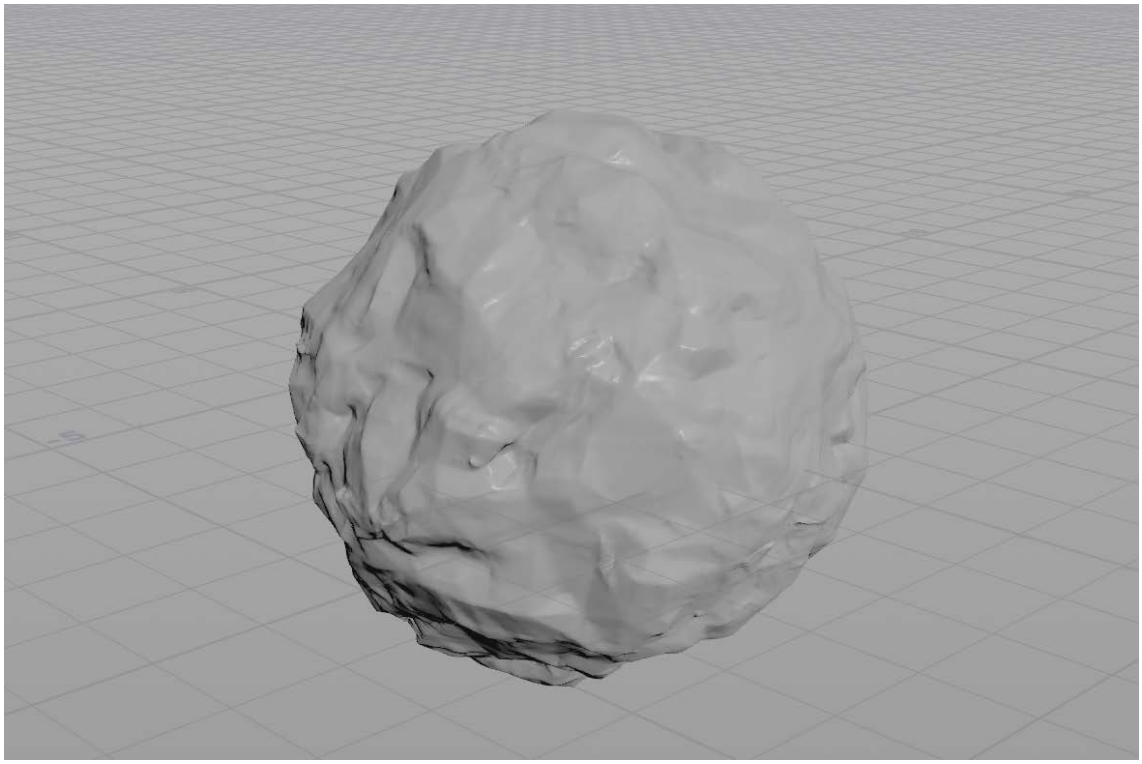

Abbildung 20 Fertige Steinstruktur mit partiell projizierten Flächen am Beispiel einer Kugel

Dieses Mesh, mit samt allen Attributen, stellt die neue noch ungefüllte Höhle dar und dient als Ausgangspunkt für alle weiteren Generierungen.

4.3 Pfade

Wie anfangs beschrieben ist die Originalgeometrie in Räume unterteilt. Von jedem dieser Räume wird nun das Zentrum ermittelt, um anhand dieser Punkte alle Räume untereinander zu verbinden.

Dazu wird innerhalb der Höhle ein gleichmäßiges, dreidimensionales Raster erzeugt. Das jeweilige Zentrum eines jeden Abschnitts wird auf den nächstliegenden Rasterpunkt projiziert. Im Anschluss wird von einem Zentrum zum nächsten der kürzeste Weg entlang des Rasters gesucht. Somit können nur Pfade erzeugt werden, die auf dem Raster liegen – sie führen also nicht durch Wände oder Decken.

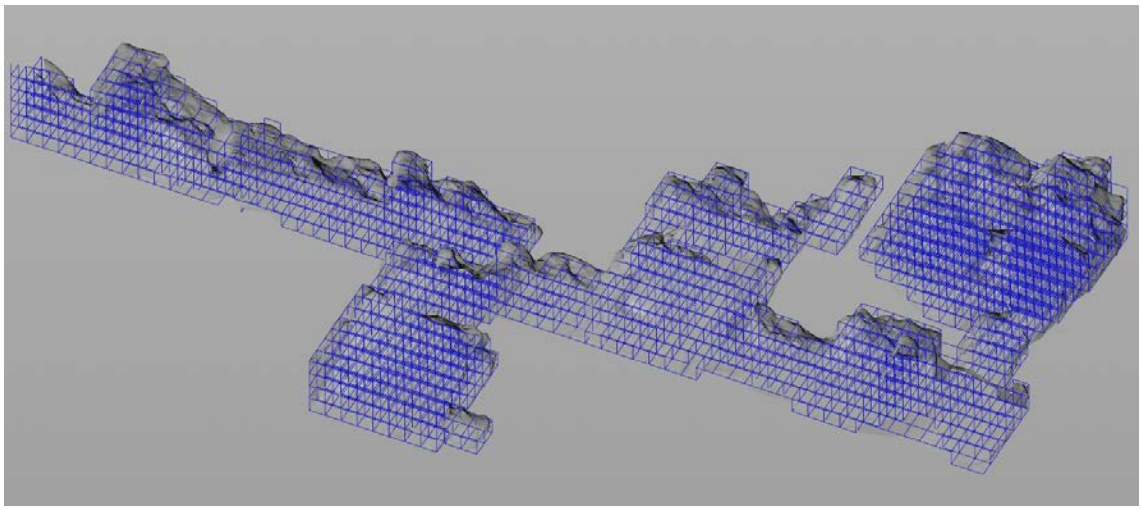

Abbildung 21 Erzeugte Raster zur Findung der Navigationspunkte im Level

Das Ergebnis ist eine Art Straßenkarte, in der die kürzesten Wege von Raum zu Raum eingezeichnet sind. Da diese Wege die Form des Rasters haben, auf dem sie erzeugt wurden, gilt es diese Map für den Spieler anzupassen. In mehreren Iterationen wird der Pfad immer wieder erneut in Richtung der abfallenden Normalen projiziert. Das bedeutet, dass der Pfad sich in die Täler der Level verschiebt.

Damit ist der Pfad erstellt und wird im Laufe der folgenden Abschnitte für die Verteilung von Vegetation (Vegetation, 4.5) und Decals (Höhle, 4.7) relevant.

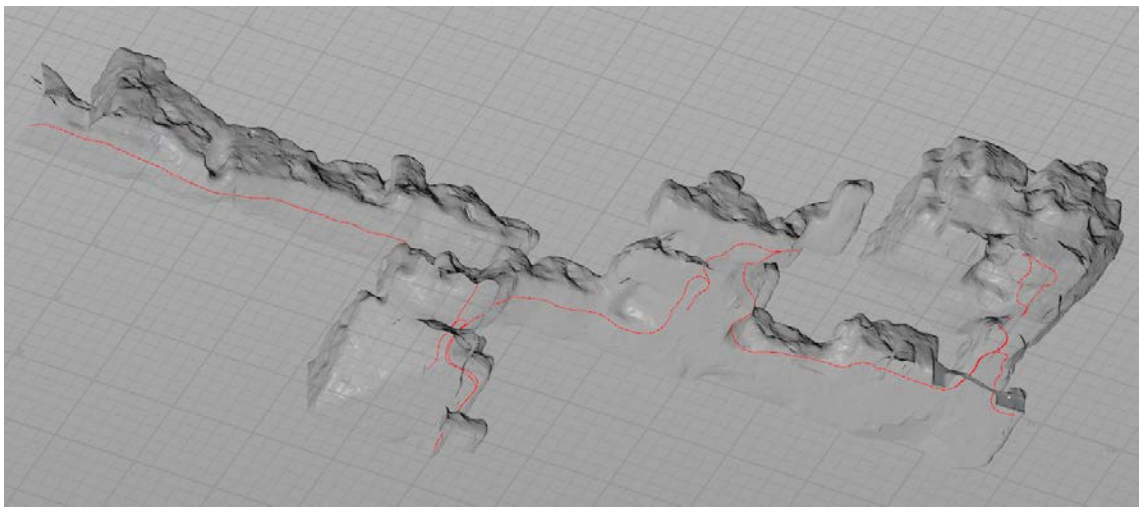

Abbildung 22 Im Level visualisierter Pfad der alle Abschnitte miteinander verbindet

4.4 Eiszapfen

Bei der Generierung der Eiszapfen gilt es visuell ansprechende Objekte zu erzeugen. Diese dürfen die Spieler*innen jedoch nicht einschränken. Zunächst wird die Geometrie auf die Flächen reduziert, auf denen Eis vorgesehen ist. Übrig bleiben vereinzelte Stellen an der Decke.

Auf der verbleibenden Geometrie werden in gleichmäßigem Abstand Punkte verteilt – diese bilden später die einzelnen Eiszapfen. Um die maximale Länge eines Eiszapfens zu bestimmen, wird ein Strahl in Richtung der negativen Y-Achse (in Richtung Boden) geschickt. Die Strecke des Strahls bis zur Kollision mit dem Boden ergibt die mögliche Maximallänge des Eiszapfens. Da die Eiszapfen somit jedoch den Weg blockieren würden muss mindestens die Größe des Spielercharakters abgezogen werden – somit wird garantiert, dass kein Eiszapfen das Gameplay behindert. Die vorerst festgelegte Größe wird am jeweiligen Punkt als Attribut gespeichert.

Nach der Bestimmung der maximalen Länge erfolgt nun die Gestaltung der Eiszapfen. Als visuelles Vorbild dient der Aufbau eines Kronleuchters. Ziel ist es die äußeren Eiszapfen möglichst kurz zu halten, so dass diese nicht die Sicht auf die in der Mitte liegenden Zapfen versperren. Im Idealfall entsteht eine Aufbau bei dem die Zapfen in der Mitte der Eisfläche am größten sind und zu den Rändern und kleiner werden.

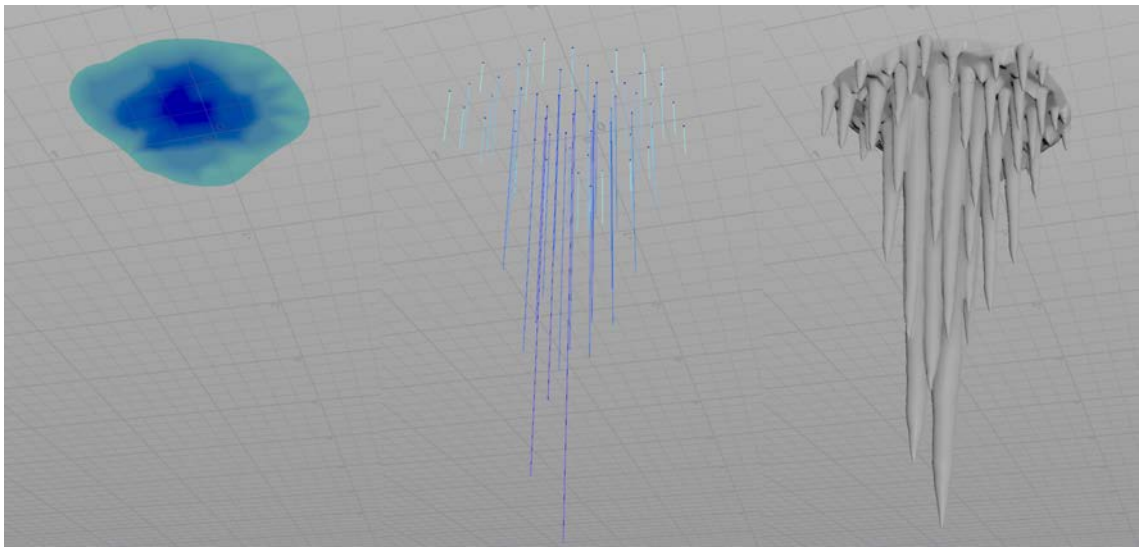

Abbildung 23 (Von links nach rechts) Visualisierter Gradient zur Skalierung der Eiszapfen; Polylines zur Prävisualisierung der Zapfen; Fertiger Zapfen nach Umwandlung in Geometrie

Dazu werden die äußeren Punkte der Eisfläche mit einem Multiplikator versehen, dem Attribut *Boundary*, welcher den Ausgangswert 1 erhält. Durch das Weichzeichnen des Attributs expandiert er auf die Nachbarpunkte, bei gleichzeitiger Verringerung des Wertes. Das Ergebnis ist ein Gradient von den äußeren Bereichen bis hin zur Mitte der Eisfläche. Wird dieser als umgekehrter Skalierungsfaktor für die zuvor gespeicherte Größe genutzt, behalten die Eiszapfen im Zentrum ihre Größe, während sie nach außen hin schrumpfen.

Auf diesen Punkten werden nun Polygone erzeugt, die sich nach unten hin verjüngen, um die typische, spitz zulaufende Form zu erhalten. Wie bereits bei der Generierung der Höhlenstruktur[44] wird die Geometrie in ein SDF umgewandelt. Durch Weichzeichnen des Volumens verschmelzen die Zapfen, die in unmittelbarer Nähe zueinander liegen.

4.5 Vegetation

Bei der Vegetation (und Steinen) handelt es sich um bereits existierende Objekte, die in dem Level verteilt werden. Dazu wird ein Punkt als Platzhalter genutzt, der später innerhalb der Unreal Engine durch eine entsprechende Instanz ausgetauscht wird.

Der Vorteil der Instanziierungen ist die Tatsache, dass das jeweilige Objekt im Grafikspeicher nur einmal vorhanden ist und jeder Punkt lediglich auf das entsprechende Objekt verweist. Bei Anwendungen wie Vegetationen, wo eine große Menge an wiederkehrenden Objekten dargestellt werden muss, ist solch eine Instanziierung nahezu unabdingbar.

Die Platzierung der Vegetation geschieht dabei durch das Attribut *Soil* (Nährboden), welches die *Density* (Dichte) steuert. Die *Density*, die die maximale Anzahl von Exemplaren auf einem Grundriss angibt, unterscheidet sich bei den verschiedenen Arten. Bei Grass ist es beispielsweise erwünscht eine höhere Dichte von Objekten zu haben – entsprechend liegt die maximale Anzahl an Objekte auf einem Grundriss bei 100

[44] Vgl. Kapitel: SDF – Signed Distance Field, 4.2.2.1

Einheiten, bei einem *Soil*-Wert von 0,5 ergeben sich 50 Einheiten. Also eine halb so dichte Menge auf dem Grundriss. Daraus ergibt sich folgende Ableitung:

Soil	**Density**
0	Keine Platzierung
0,5	Auf der Grundfläche wird die Hälfte der maximalen Anzahl platziert
1	Maximale Anzahl wird auf der Grundfläche platziert

Der Ausgangswert des *Soils* beträgt Anfangs 0,5 – somit ist die halbe Kapazität vorgesehen. In den darauffolgenden Schritten wird dieser Wert nach und nach durch verschiedene Faktoren beeinflusst. Zum einen durch die Attribute, welche der Geometrie direkt angehangen wurden und zum anderen wird auch der Pfad zur Berechnung herangezogen, um ein Durchkommen der Spieler*innen, sowie für eine bessere Führung durchs Level zu gewährleisten.

Möglich Faktoren, die den *Soil*-Wert beeinflussen sind:

Einfluss	**Beschreibung**
-1,0	Schnee
-0,3	Pfad
-0,1	Ziegel
+0,5	Pflanzen

Abbildung 24 Auszug der Steine und Pflanzen die verwendet werden[45]

Um Ausnahmefälle zu steuern, beispielsweise für die Platzierung von sehr großen Steinen, können die Werte des *Soils* zusätzlich objektspezifisch angepasst werden. Somit ist gewährleistet das große Felsen auf keinen Fall auf Wegen platziert werden, Grass jedoch zu einem gewissen Anteil dort entstehen kann.

Da die Temperatur in der Höhle lediglich anhand der Typen rekonstruiert wird ergeben sich Ungenauigkeiten. Diese Ungenauigkeiten haben beispielsweise zur Auswirkung, dass an Stellen an denen Eiszapfen vorgesehen wären eine zu hohe Temperatur ermittelt wurde, um diese zu platzieren. Als visuelle Alternative werden auf Basis der Eiszapfen an diesen Stellen Pflanzen entlang der Höhlendecke platziert werden.

Um später in der *Unreal Engine* einen Windeffekt darstellen zu können, wird ein Gradient über die gesamte Y-Achse entlang erstellt. Zusammen mit der Temperatur wird diese Information in den Vertex-Color festgehalten, um später in der *Unreal Engine* mit diesen Informationen arbeiten zu können[46].

---

[45] Megascans Assets: https://quixel.com/megascans/library/latest
[46] Vgl. Kapitel: Vertex Color, 4.8.2

4.6 Ziegel

Bei der Generierung der Ziegel ergibt sich (ähnlich dem Boden) die Eigenschaft, dass beide nur minimal bis gar nicht deformiert wurden. Dies hat beim Boden zum einen Gameplay-technische Gründe. Bei den Ziegeln sind es jedoch optische, da die Ziegel auf einer mehr oder weniger ebenen Fläche aufgebracht werden sollen um ein gradliniges Erscheinungsbild beizubehalten.

Daraus ergibt sich die Tatsache, dass die Original-Polygone für die Ziegel fast deckungsgleich mit den vorgesehen Stellen in der neuen Höhle sind.

Statt die Ziegel zufällig auf der entsprechenden Fläche zu verteilen, sollen sie in dem für ein Mauerwerk typischen versetzen Muster platziert werden. Dies bedeutet, dass die ungerade Reihe zur geraden jeweils um ungefähr die Hälfte versetzt ist.

Als Ausgangspunkt dient hierbei nicht wie bei den anderen Komponenten die neu generierte Höhle, sondern das Original Mesh vor der Deformation.

Hierbei wird es sich zu Nutze gemacht, dass die Polygone des alten Levels bereits UV-Koordinaten besitzen. Die UVs der zusammenhängenden Polygone müssen lediglich verbunden werden, damit sich für zusammenhängende Flächen auch zusammenhängende UV-Patches ergeben.

Die Polygone werden daraufhin anhand ihrer UV-Koordinaten ins Welt-Koordinatensystem (X- und Z-Achse, Y ausgenommen) projiziert. Deckungsgleich wird in entsprechender Skalierung die Ziegelstruktur über diese Polygone gelegt.

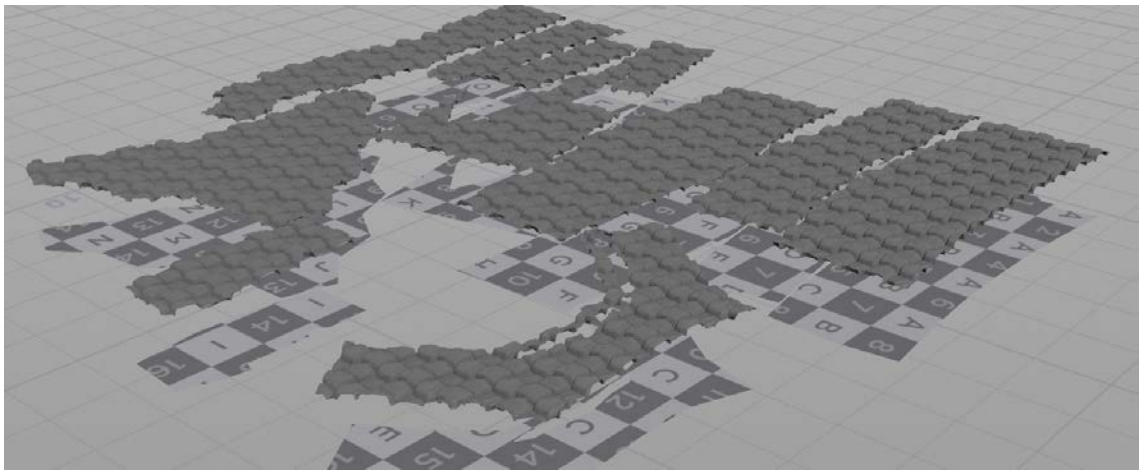

Abbildung 25 Geometrie der Ziegel und darunter liegende Originalgeometrie mit Checkerboard, zur Visualisierung der UV-Koordinaten

Mit Hilfe eines Strahls auf die darunter liegenden Polygone können rückwirkend über die UV-Koordinate die vorherigen Welt-Koordinaten ermittelt werden. Wodurch die Ziegelstruktur sich gegen das alte Mesh austauschen lässt.

4.7 Höhle

Als Zentrum des prozeduralen Systems diente bisher die generierte Höhle[47], mit ihren Attributen. Diese Attribute werden nun entfernt, da sie lediglich als Grundlage für die anderen Module dienten, ausgenommen der *Ground* und *Temperature* Attribute. Diese Attribute werden zur Gestaltung des Materials in die Vertex-Color geschrieben und zusätzlich um eine Ambient Occlusion Map ergänzt.

Entsprechend dem Original werden Fußspuren im Schnee gesetzt. Dazu wird der generierte Pfad[48] genutzt und auf die Bereiche, an dem später Schnee erscheint, gekürzt. Entlang des Pfades werden in unregelmäßigen Abständen einfache quadratische Polygone erzeugt, die auf den Boden projiziert werden.

---

[47] Vgl. Kapitel: Aufbereitung, 4.2

[48] Vgl. Kapitel: Pfade, 4.3

In der Unreal Engine werden diese simplen Polygone mit Decals belegt, welche die Fußspuren auf die darunter liegende Textur projizieren.

Abbildung 26 Vergleich der Fußspuren aus Tomb Raider und der des prozeduralen Systems

4.8 Materials

Die Geometrie wird in *Unreal* mit verschiedenen Materials versehen, die alle nach dem gleichen Schema aufgebaut sind. Den einzigen Unterschied stellen vorhandene bzw. nicht vorhandenen UV-Koordinaten da.

4.8.1 Triplanares Mapping

Da die Pflanzen und Steine (nicht die Höhle selbst) bereits UV-Koordianten besitzen, sind diese vom Triplanaren Mapping ausgenommen – für jegliche andere Geometrie werden auf diese Weise Textur-Koordinaten erzeugt.
Bei Triplanarem Mapping wird die Textur anhand der Welt-Koordinaten (X-,Y- und Z-Achse) auf dem Objekt platziert und anhand der Richtung der Oberflächennormalen verblendet [32].

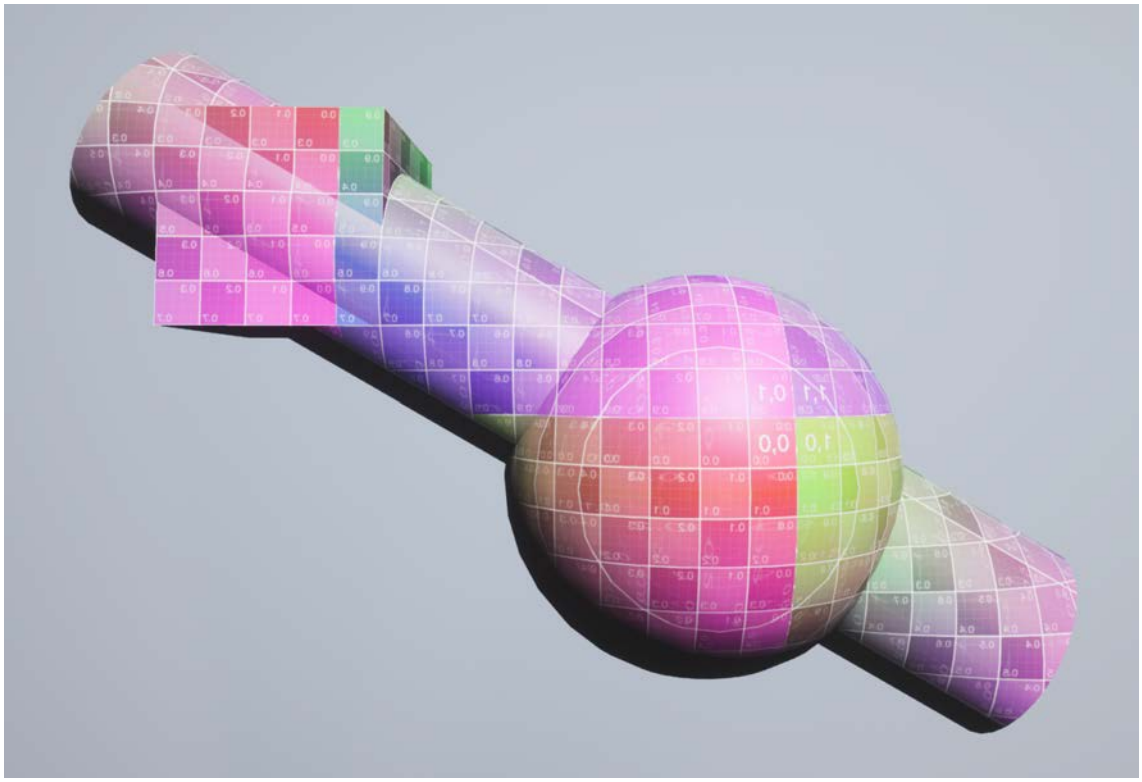

Abbildung 27 Visualisierung des Triplanaren Mappings anhand eines Checkerboards und drei unterschiedlichen Objekten ohne UV-Koordinaten

Da die UV-Koordinaten anhand des Welt-Koordinatensystems erzeugt werden ergeben sich auch zwischen getrennten Objekten fließende Übergänge. Mit dieser Methode lassen sich Schnee und Moos ohne auffallende Übergänge übergreifend auf die Objekte projizieren.

4.8.2 Vertex Color

Im Höhlensystem existieren unterschiedliche Klimazonen, dabei gibt es für jedes Objekt (Höhle, Pflanzen, Steine) jeweils nur ein Material, das in der Lage ist, die unterschiedlichen Temperaturen abzubilden.

4.8.2.1 Blau-Kanal

Der Blau-Kanal der Vertex Color (RGB) enthält bei allen Objekten Informationen zur ihrer Umgebungstemperatur. Anhand dieses Wertes kann das Objekt mit Moos oder Schnee bedeckt sein. Bei einem neutralen Wert von 0,5 wird kein zusätzlicher Layer eingeblendet. Wird der Wert nach oben oder unten verschoben, wird die Deckkraft für den jeweiligen Layer angepasst.

Die entsprechenden Wirkungsbereiche der Klimazonen gestalten sich wie folgt:

Schnee	**Neutral**	**Moos**
0-0,4	0,5-0,6	0,7-1

Die anderen Kanäle (R und G) stehen zur freien Verfügung für objektspezifische Eigenschaften.

Abbildung 28 Objekte sind mit Schnee bedeckt, Blau-Wert der Vertex-Color liegt bei 0

Abbildung 29Objekte im neutralen Zustand. Blau-Wert der Vertex-Color liegt bei 0,5.

Abbildung 30 Objekte sind mit Moos bedeckt, Blau-Wert der Vertex-Color liegt bei 1

4.8.2.2 Rot- und Grün-Kanal für die Höhle

Im Rot-Kanal der Höhle befindet sich eine eingebackene Ambient Occlusion Map, um eine deutlichere Umgebungsverdeckung zu erzeugen. Im Grün-Kanal befindet sich eine Maske für den Untergrund, damit dieser nicht mit derselben Steinstruktur wie Decke und Wand versehen wird.

4.8.2.3 Rot-Kanal bei Pflanzen

Bei den Pflanzen dient der Rot-Kanal für einen Gradienten entlang der Y-Achse. Dieser wird als Multiplikator für einen Pixel-Offset genutzt, mit dem die Windbewegung der Pflanzen simuliert wird. In der Nähe des Bodens besitzt der Gradient den Wert O und in Richtung der Blätter 1. Somit bleibt der Stamm der Pflanze im Boden verankert und bewegt sich nicht mit.

5 Ergebnis

Die nachfolgenden Seiten zeigen eine Gegenüberstellungen des Originallevels und der Ergebnisse des prozeduralen Systems, in chronologischer Reihenfolge des Levelverlaufs.

5.1 Gegenüberstellung

Abbildung 31 Vergleich von Originallevel und prozeduralen Ergebnis (1)

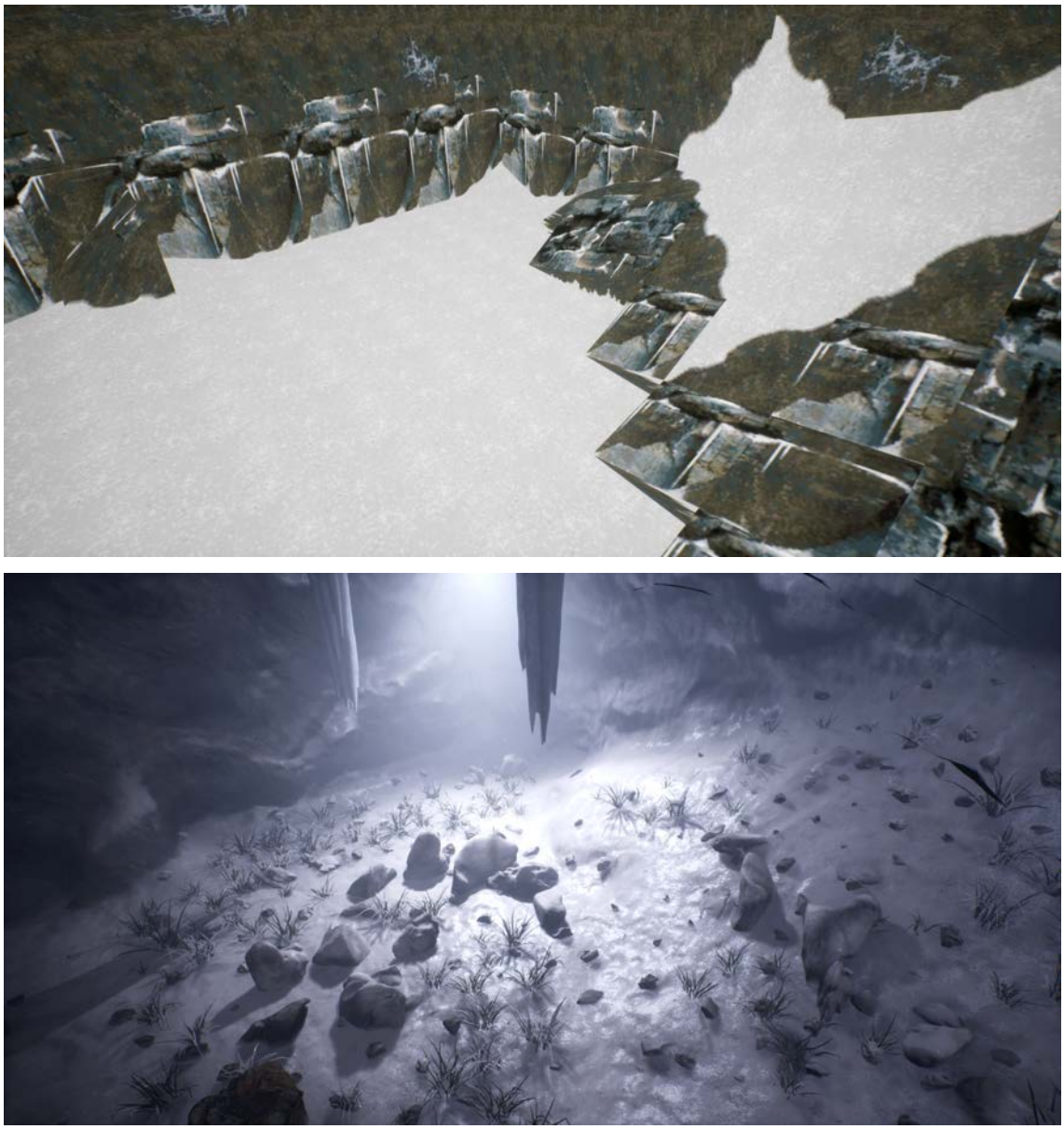

Abbildung 32 Vergleich von Originallevel und prozeduralen Ergebnis (2)

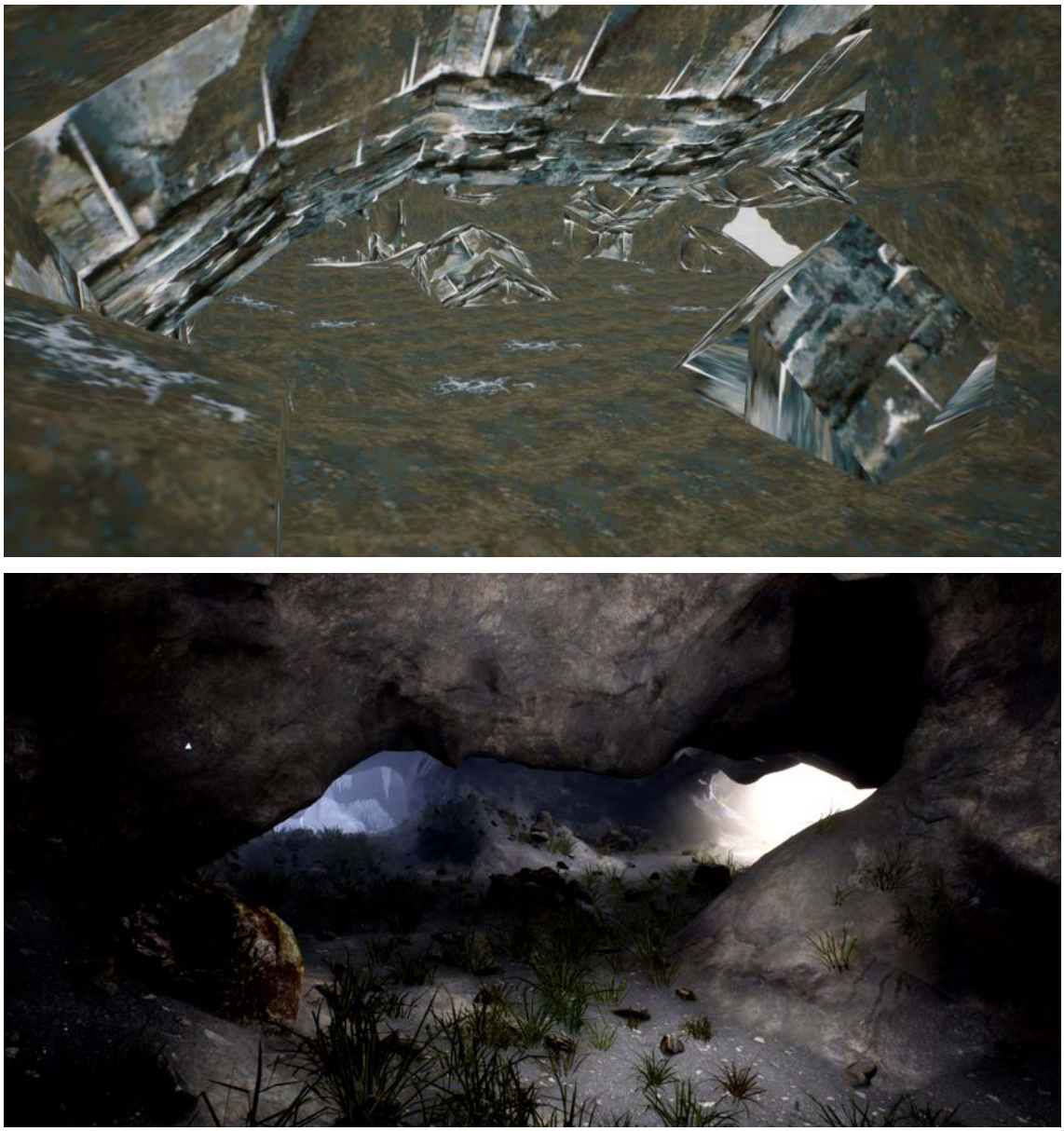

Abbildung 33 Vergleich von Originallevel und prozeduralen Ergebnis (3)

Abbildung 34 Vergleich von Originallevel und prozeduralen Ergebnis (4)

Abbildung 35 Vergleich von Originallevel und prozeduralen Ergebnis (5)

Abbildung 36 Vergleich von Originallevel und prozeduralen Ergebnis (6)

5.2 Variationen beim Temperaturwert

Abbildung 37 Prozedurale Ergebnisse bei unterschiedlicher Temperatur (1)

Abbildung 38Prozedurale Ergebnisse bei unterschiedlicher Temperatur (2)

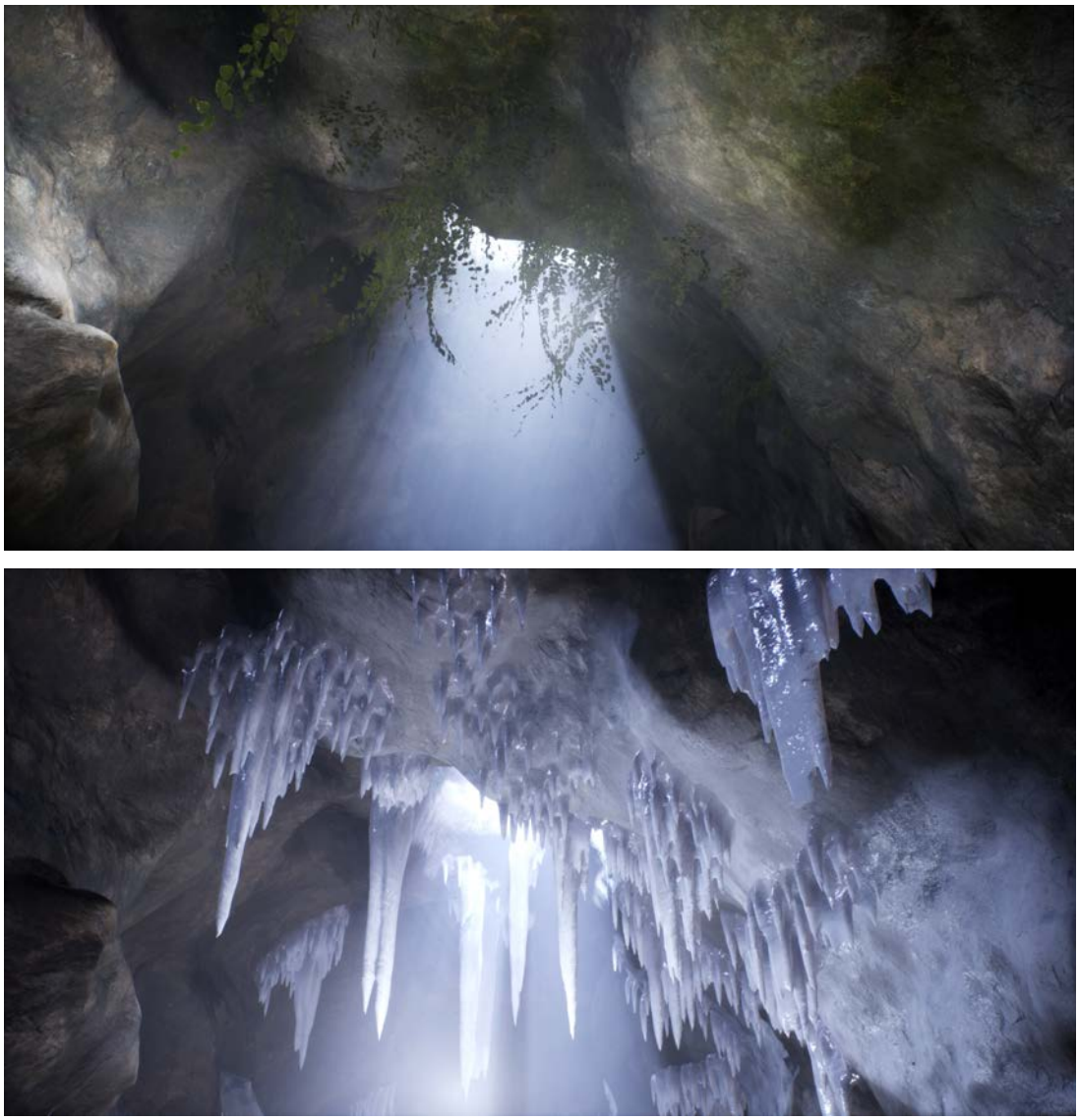

Abbildung 39Prozedurale Ergebnisse bei unterschiedlicher Temperatur (3)

Abbildung 40Prozedurale Ergebnisse bei unterschiedlicher Temperatur (4)

Abbildung 41Prozedurale Ergebnisse bei unterschiedlicher Temperatur (5)

5.3 Zusammenfassung

Zu Beginn der Entwicklung des prozeduralen Systems ging es hauptsächlich um das Entwickeln, Umsetzen und Verwerfen von ersten Konzepten. Manchen Ansätze zeigten ihre Defizite bereits auf dem Papier, andere wiederum erst im Laufe der Umsetzung.

Jedes dieser frühen Konzepte führte jedoch zu neuen Erkenntnissen, die letztlich zu dem hier dokumentierten Ansatz führten: Dem Konvertieren in ein temporäres Volumen und dem Aufbau in Modulen. Auch bei den einzelnen Modulen war der hier festgehaltene Ansatz nicht immer die erste Idee.

Die benötigte Zeit für die Entwicklung solcher Konzepte ist ein maßgeblicher Bestandteil bei der prozeduralen Generierung. Wie bereits zu Anfang zitiert, sagte Frieder Nake dazu:

> *„Generative Gestaltung zielt auf die Unterfläche und zeigt sich auf der Oberfläche“ [6]*

Bis jedoch die Unterfläche nach oben gelangt, dauert es eine Zeit – eine Zeit in der Lösungsansätze entwickelt und erste Ideen umgesetzt werden. Im Vergleich zum direkten Modeling entsteht somit erst später ein visueller Eindruck. Dennoch benötigt es diese Zeit und bildet maßgeblich das Fundament des späteren prozeduralen Systems.

Durch das Separieren in unterschiedliche Module war es möglich, für die einzelnen Aufgaben unterschiedliche Methoden zu entwickeln. Jedem Modul steht dieselbe Grundlage zur Verfügung und sie stehen in keiner bedeutenden Abhängigkeit zueinander. Somit lassen sich Fehler in der Generierung gezielt lokalisieren.

Nach dem Fertigstellen der einzelnen Module lässt sich ein Wendepunkt erkennen, bei dem es sich die maßgeblichen Vorteile der prozeduralen Generierung gegenüber dem klassischen Modeling zeigen.

Nicht nur dadurch, dass das System automatisiert auf Änderungen der Level reagiert – es kommt auch zu einem früheren Polishing und ermöglicht entsprechend mehr Iterationen. Statt jedes einzelne Objekt von Hand zu bearbeiten, wird das Regelwerk angepasst, welches daraufhin in wenigen Sekunden[49] das überarbeitete Ergebnis liefert. Ganz gleich, ob es sich dabei um das Platzieren von 10.000[50] Pflanzen oder das Erstellen der gesamten Höhlenstruktur handelt.

Während sich bei dem Generieren von großen Daten die Stärken des prozeduralen Workflows gezeigt haben, ergeben sich Nachteile bei der Erstellung von sogenannten Hero Assets. Allgemein werden einzigartige Objekte als Hero Assets bezeichnet. Im Kontext des Level Designs kann dies aber auch ein besonderes Umgebungsmerkmal sein. Ein solches Umgebungsmerkmal könnte z.B. die Säule in der Mitte der Höhle darstellen[51]. Hierbei lassen sich hervorstechende Umgebungsmerkmale erzeugen, indem ihre Regeln seltener angewendet werden.

Da es jedoch in der Regel das Ziel ist eine besondere Eigenschaft hervorzuheben, wie z.B. die genannte Steinsäule, benötigt es ein gezieltes und komplexes Regelwerk, welches ein einziges Mal angewendet wird. Demnach kann das klassische Modeling hierbei zu einem schnelleren (und unter Umständen auch besseren) Ergebnis führen.

In Anbetracht der eigentlichen Aufgabe, dem prozeduralen Generieren eines Levels auf Basis eines älteren Vorbilds, lässt sich folgendes zusammenfassen: Beim Vergleich des Originallevels mit dem prozedural generierten Ergebnis lässt sich feststellen, dass die Proportionen und der Aufbau des Originals stets beibehalten wurden. Eiszapfen, Pflanzen und Ziegelstrukturen wurden an den entsprechenden Stellen erzeugt, sofern es unter den ermittelten Bedingungen möglich war.

---

[49] Das Durchlaufen des Algorithmus benötigte mit zunehmender Komplexität auch mehr Zeit. Von wenigen Sekunden, bis später zu 10 Minuten, für eine komplette Generierung. Durch die Aufteilung in Modulen ist eine komplette Generierung jedoch nur äußert selten notwendig. Je nach Modul lag die Dauer bei 30 Sekunden bis hin zu 3 Minuten.

[50] Im fertigen Level werden über 23.000 Objekte platziert.

[51] Vgl. Bild auf Seite 67

Obwohl das geschaffene Regelwerk relativ übersichtlich ist, kam es an kritischen Stellen zu keinerlei Problemen bei der Generierung und das Ergebnis war stets zufriedenstellend. Sobald die Typisierung der Texturblöcke abgeschlossen war, kam das System ohne weiteres Eingreifen eines Designers oder einer Designerin zum gewünschten Ergebnis.

5.4 Ausblick

Um das entwickelte prozedurale System künftig auch für andere Level abseits der *Tomb Raider* Reihe nutzbar zu machen, werden Änderungen notwendig sein. Die Typisierung anhand der UV-Koordinaten funktioniert innerhalb von Games, die ähnlich des *Tomb Raider* Levels aufgebaut sind, auf Grund der vorgegebenen Strukturen sehr gut. Moderne Level nutzen selten einen Textur Atlas, der sich auf diese Weise kategorisieren lässt. Dieses Problem lässt sich umgehen, indem die Attribute auf das Mesh per Pinsel[52] aufgetragen werden. Diese Methode ist im Vergleich zum Anlegen der Typen arbeitsintensiver, lässt sich aber auch beim Erstellen von neuen Leveln[53] anwenden – wodurch der Einsatzbereich des Systems deutlich erweitert wird.

Im Zuge dieser Arbeit wurde das entwickelte System als 1-Klick Lösung konzipiert. Dies geschah auf Basis der Fragestellung, wie weit ein prozedurales System in der Lage ist, autark Ergebnisse zu liefern. Für einen produktiven Einsatz sollten die Module die Möglichkeit besitzen, differenzierter beeinflusst zu werden. Derzeitig können punktuelle Änderungen in allen Teilen des Systems weitreichende Folgen haben. So eine Automatik kann in den ersten Iterationen zur Stilfindung hilfreich sein, im Anschluss lässt sich das gewünschte Ergebnis allerdings nur umständlich formen.

Das in dieser Arbeit entwickelte System könnte in Zukunft bei künftigen Arbeiten zur Anwendung kommen. Dies ist im Hinblick auf das Zusammenspiel aller oder aber auch einzelner Module denkbar.

---

[52] Dazu existiert in *Houdini* die *Paint Node*. Mit dieser lassen sich Attribute auf das Mesh zeichnen: https://www.sidefx.com/docs/houdini/nodes/sop/paint.html

[53] Vgl. Kapitel: Blocking, 3.1.1

6 Abbildungsverzeichnis

7 Literaturverzeichnis

[1] A. Amato, „*Procedural Content Generation in the Game Industry*“ in Game Dynamics, Springer, 2017.

[2] P. Galanter, „*What is Generative Art? Complexity Theory as a Context for Art Theory*“, 2003.

[3] „*PCG Wiki – Games Featuring Procedural Generation*“, Online: http://pcg.wikidot.com/category-pcg-games [Zugriff am 20.05.2019].

[4] G. N. Yannakakis, J. Togelius, „*Experience-Driven Procedural Content Generation*“, 2011.

[5] J. Togelius, E. Kastbjerg, D. C. Shedl, G. Yannakakis, „*What is procedural content generation?: Mario on the borderline*“, PCGames '11, 2011.

[6] Digitale Medien Bremen, „*Zeichen oder Rechnen: Der Computer ist Dabei*“, Online: https://vimeo.com/23631676 [Zugriff am 16.1.2019].

[7] P. Prusinkiewicz, A. Lindenmayer, „*The Algorithmic Beauty of Plants*“, New York: Springer-Verlag New York, 1990.

[8] G. Smith, „*The Future of Procedural Content Generation in Games*“, 2014.

[9] K. Xu, D. Campeanuy, „*Houdini Engine: Evolution Towards a Procedural Pipeline*“, 2014.

[10] M. Hendrikx, S. Meijer, J. Van Der Veldeb, A. Iosup, „*Procedural Content Generation for Games: A Survey*“, Netherlands: ACM Trans. Multimedia Comput. Commun., 2011.

[11] „*Tate – Frieder Nake*“, Online: https://www.tate.org.uk/art/artists/frieder-nake-17874 [Zugriff am 23.05.2019].

[12] „*Spalter Digital – Hall (Corridor)*“, Online: http://spalterdigital.com/artworks/computergrafik-computerplastik-series-7/ [Zugriff am 20.05.2019].

[13] CRPG Addict, „*The CRPG Addict: Game 79: Beneath Apple Manor (1978)*“, Online: http://crpgaddict.blogspot.com/2012/12/game-79-beneath-apple-manor-1978.html [Zugriff am 17.1.2019].

[14] „*Wikipedia – Rogue (Computerspiel)*“, Online: https://de.wikipedia.org/wiki/Rogue_(Computerspiel) [Zugriff am 23.03.2019].

[15] „*fxguide – Side Effects Software – 25 years on*“, Online: https://www.fxguide.com/featured/side-effects-software-25-years-on/ [Zugriff am 20.05.2019].

[16] Assembly Seminars, „*Kkrieger content creation in 96kb - Farbrausch*“, Online: https://www.youtube.com/watch?v=uUfV41HE_Dg [Zugriff am 16.1.2019].

[17] „*Processing*“, Online: https://processing.org/overview/ [Zugriff am 10.05.2019].

[18] Maxis, „*Spore Creature Editor*“, Online: http://www.spore.com/creatureCreator [Zugriff am 15.1.2019].

[19] „*Spore*“, Online: http://www.spore.com/what/spore [Zugriff am 23.03.2019].

[20] A. Doull, „*The Death of the Level Designer*“, 2008.

[21] G. Chen, G. Esch, P. Wonka, P. Müller, E. Zhang, „*Interactive Procedural Street Modeling*“, 2008.

[22] N. Shaker, J. Togelius, M. J. Nelson, „*Procedural Content Generation in Games*“, Springer, 2016.

[23] „*A Behind-The-Scenes Tour Of No Man's Sky's Technology*“, Game Informer, Online: https://www.youtube.com/watch?v=h-kifCYToAU [Zugriff am 11.1.2019].

[24] „*No Man Sky*“, Online: https://www.nomanssky.com/press/ [Zugriff am 03.05.2019].

[25] G. Smith, „*Understanding procedural content generation: A design-centric analysis of the role of PCG in games*“, 2014.

[26] „*You Tube – 20 Minutes of Far Cry 5 Fly, Fishing, and Killing Gameplay in 4K - PSX 2017*“, Online: https://www.youtube.com/watch?v=STg-lGNo5sk [Zugriff am 24.05.2019].

[27] „*YouTube – Procedural World Generation of Ubisoft's Far Cry 5 | Etienne Carrier | Houdini HIVE Utrecht*“ Online: https://www.youtube.com/watch?v=NfizT369g60 [Zugriff am 22.05.2019].

[28] „*Supyrb – Marbloid Presskit*“, Online: http://www.supyrb.com/presskit/marbloid/ [Zugriff am 8.05.2019].

[29] F. Hoekstra, „*Graduation report: Procedural environmental design*“, 2011.

[30] „*Steam – Tomb Raider I*“ Online: https://store.steampowered.com/app/224960/Tomb_Raider_1/ [Zugriff am 10.02.2019]

[31] „*2019 Houdini Games Reel*“ Online: https://vimeo.com/323879609 [Zugriff am 22.05.2019].

[32] „*Martin Palko – Triplanar Mapping*“ Online: http://www.martinpalko.com/triplanar-mapping/ [Zugriff am 23.05.2019].

[33] J. Togelous, G. N. Yannakakis, K. O. Stanley, C. Browne, „*Search-Based Procedural Content Generation: A Taxonomy and Survey*“, Transactions on Computational Intelligence and Ai in Games, 2011.